January 2016

Energy-Efficient System Architectures for Intermittently-Powered IoT Devices

Hrishikesh Jayakumar
Purdue University

Graduate School Form 30
(Revised 08/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Hrishikesh Jayakumar

Entitled Energy-Efficient System Architectures for Intermittently-Powered IoT Devices

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

VIJAY RAGHUNATHAN

ANAND RAGHUNATHAN

BYUNGHOO JUNG

KAUSHIK ROY

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

VIJAY RAGHUNATHAN

Approved by Major Professor(s): _____

Approved by: V. Balakrishnan 07/26/2016

Head of the Department Graduate Program Date

ENERGY-EFFICIENT SYSTEM ARCHITECTURES FOR INTERMITTENTLY-POWERED IoT DEVICES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Hrishikesh Jayakumar

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2016

Purdue University

West Lafayette, Indiana

Dedicated to my parents Lekha & Jayakumar, for encouraging me to dream beyond and to my wife Praveena, for believing in me

ACKNOWLEDGMENTS

An incredible number of people have supported and helped me in the successful completion of this dissertation and through this phase of my life. First and foremost, I want to thank my adviser **Prof. Vijay Raghunathan** for his careful and conscientious guidance, invaluable support, and constant encouragement throughout the course of my Ph.D. His ability to quickly perceive ideas at the most relevant higher abstraction without getting mired in details, and choose the most apt words to express it has always amazed me and has further motivated me to constantly improve. I will forever be grateful for the umpteen discussions that I had with him over research, his perspectives on future directions in technology, and on day-to-day issues. My research discussions with him have always been stimulating and equally thought-provoking for me. I want to express my appreciation and sincere gratitude to him for going over and beyond – putting in the long hours, on short notice, to sharpen and strengthen my skills in the preparation of each manuscript and presentation, often despite his own personal and other professional commitments – day or night, weekdays and weekends alike. I truly appreciate the effort that he put in when I was going through challenging personal circumstances, and in motivating me and lifting my confidence when I was down. His exemplary decision making ability and uncompromising support for his own even in the most adverse of scenarios are qualities that I seek to emulate in the future. I respect him for providing me with the room for failure and in retrospection, acknowledge him for molding me from the raw enthusiast I was into the mature researcher I am. I am extremely honored to have worked with an outstanding guide as him during the course of my Ph.D. and I am confident that the thought processes and skills that he has inculcated in me in the past few years will hold me in good stead for the rest of my career.

I also want to thank the rest of my dissertation committee,

Prof. Anand Raghunathan: For the multiple research group discussions, which were real eye-openers as far as critiquing papers go, for pairing me with Vinay in the project which turned out to be my first-ever publication, and for lending the tool that increased the scope of measurements in all my publications.

Prof. Kaushik Roy: For providing me the platform for a brief flirtation with Quantum Computing. It has left me yearning for more and I am sure to pursue it at a later point in my career.

Prof. Byunghoo Jung: For providing constructive feedbacks during the preliminary and final exams.

Pursuing a Ph.D. can be a very lonely experience. In that, I want to sincerely thank my lab-mate, Arnab for bringing an abundance of positivity and enthusiasm to the Embedded Systems Laboratory (ESL). Altogether, I have collaborated with him for twelve papers in four years and it has been nothing but quite an astonishing partnership. He has also been my chief sparring partner during the course of my stay at Purdue, in terms of views – technical, philosophical, and otherwise. I am grateful to know Woo Suk and thank him for the various interactions, research collaborations, and especially, for teaching me the fundamentals of PCB routing and assembly. Working with him has been an absolute joy and I am grateful to him, in particular, for introducing me to Korean cuisine. I am fortunate to know Younghyun Kim and thank him for his readiness to review manuscript drafts on demand, and for sharing his views on the broader topic of relevant and impactful research. Additionally, I also thank other ESL members – alumni and current, Alim Al Islam, Sajjad, Kangwoo, and Soubhagya for the very interesting and sometimes unconventional discussions that we had. I would also like to thank the members of Integrated Systems Laboratory for their friendship – especially, Vivek, Rangha, Vinay, Yue, Swagath, Shankar, Ashish, and Shubham. I would also like to thank my friends Venkat, Gangi, Vijay Mantha, Prateek, Abhisek, Chinna, Deepan, Sridhar, Anup and the rest of the gang

for adding color to my life outside the lab. All of you will always remain close to my heart. Thank You! The pleasure is all mine.

Finally, I owe this dissertation and the utmost gratitude to my wife, Praveena, who waited a long time. I will always be indebted to her for shouldering the entire responsibility of the family during this phase of my life and I cannot thank her enough for her unwavering support even in the most challenging of circumstances. I would like to thank my daughter, Advaita, whose mere presence and smile brought joy to me. I also want to thank my parents, Lekha and Jayakumar, and my sister, Haripriya, whose support, sacrifice, and love have helped me reach where I am today. I dedicate my dissertation to them.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ADC	Analog to Digital Converter
BLE	Bluetooth Low Energy
COTS	Commercially off the shelf
DRV	Data Retention Voltage
eNVM	emerging Non-volatile Memory
FeRAM	Ferroelectric Random Access Memory
GPR	General Purpose Register
IoT	Internet of Things
ISR	Interrupt Service Routine
MCU	Microcontroller
MRAM	Magnetoresistive Random Access Memory
NVM	Non-volatile Memory
PC	Program Counter
RTC	Real Time Clock
SP	Stack Pointer
SR	Status Register
SRAM	Static Random Access Memory
SVS	Supply Voltage Supervisor

ABSTRACT

Jayakumar, Hrishikesh PhD, Purdue University, August 2016. Energy-Efficient System Architectures for Intermittently-Powered IoT Devices. Major Professor: Vijay Raghunathan.

Various industry forecasts project that, by 2020, there will be around 50 billion devices connected to the Internet of Things (IoT), helping to engineer new solutions to societal-scale problems such as healthcare, energy conservation, transportation, *etc.* Most of these devices will be wireless due to the expense, inconvenience, or in some cases, the sheer infeasibility of wiring them. With no cord for power and limited space for a battery, powering these devices for operating in a *set-and-forget* mode (*i.e.*, achieve several months to possibly years of unattended operation) becomes a daunting challenge. Environmental energy harvesting (where the system powers itself using energy that it scavenges from its operating environment) has been shown to be a promising and viable option for powering these IoT devices. However, ambient energy sources (such as vibration, wind, RF signals) are often minuscule, unreliable, and *intermittent* in nature, which can lead to frequent intervals of power loss. Performing computations reliably in the face of such power supply interruptions is challenging.

Intermittently-powered IoT devices are an emerging class of embedded devices that operate on energy harvested from intermittent sources. These devices execute long running programs incrementally (in small steps each power-`ON` period) and across multiple power-`ON` periods. A prerequisite for operating in this manner is the need for some form of checkpointing of system state from SRAM to non-volatile memory when power loss is imminent. Traditionally, microcontrollers have employed Flash memory as the primary non-volatile storage technology. However, the energy (and latency) intensive operations of Flash make it inefficient for frequent checkpointing,

and consume a significant amount of energy that could otherwise be used for executing meaningful application-related computations and tasks.

This dissertation proposes system architectures to improve the energy-efficiency of intermittently-powered IoT devices while ensuring the reliability and forward progress of applications executing on them. First, to reduce the checkpoint overhead, we explore a unified memory architecture using an emerging non-volatile memory. Recent advances in memory technology has resulted in the emergence of non-volatile memory that combine the benefits of SRAM with the non-volatility of Flash. Memories such as Ferroelectric RAM (FeRAM), Magnetoresistive RAM (MRAM), *etc.*, have superior power-performance characteristics, as compared to Flash. In this dissertation, we propose an *in-situ* checkpointing scheme using a unified-FeRAM architecture to reduce the checkpointing overhead and demonstrate that it enables the efficient usage of gathered energy. Second, we present an energy-aware dynamic memory mapping scheme for hybrid FeRAM-SRAM MCUs in intermittently-powered IoT devices to exploit both the reliability benefits of FeRAM and the performance benefits of SRAM. Even though FeRAM is non-volatile, it is slower than SRAM and have a higher power consumption. However, SRAM is volatile making it unreliable for intermittently-powered IoT devices. Hence, in this dissertation, we propose an intermediate approach in hybrid FeRAM-SRAM MCUs to benefit from the non-volatility of FeRAM and the speed of SRAM. Last, we architect a new low power mode for deeply embedded MCUs by performing sleep mode voltage scaling to enable SRAM data retention at ultra-low power consumption. Most IoT devices operate in an intermittent manner wherein they become active for a short duration of time to perform the intended task and then enter a sleep mode. However, present day sleep modes of MCUs are energy-inefficient due to the requirement of retaining state. Hence, we propose a new low power sleep mode that retains the SRAM data at ultra-low power consumption and demonstrate the powering of the proposed mode via harvesting minuscule amounts of ambient energy.

We believe that the contributions made in this dissertation take a significant step in realizing *set-and-forget* IoT devices and in furthering the field of intermittently-powered computing.

1. INTRODUCTION

The Internet of Things (IoT) is expected to pervade all aspects of human life and fundamentally alter the way we interact with our physical environment, helping to envision and engineer new solutions to a variety of societal-scale problems such as healthcare, home automation, energy conservation, asset tracking, maintenance of public infrastructure, *etc.* (as shown in Fig. 1.1). The IoT machinery to realize this vision is anticipated to be composed of electronic devices performing the distinct yet complementary functionalities of sensing physical phenomena, processing the gathered data, and relaying data in-between the processing and sensing frameworks. Various industry forecasts predict an exponential increase in the number of devices that will be deployed for the IoT, with some forecasts projecting a total of about 50 billion [1] devices by the year 2020 (see inset in Fig. 1.1). Fig. 1.2 illustrates the three distinct types of devices that compose the IoT, represented as a hierarchy of three

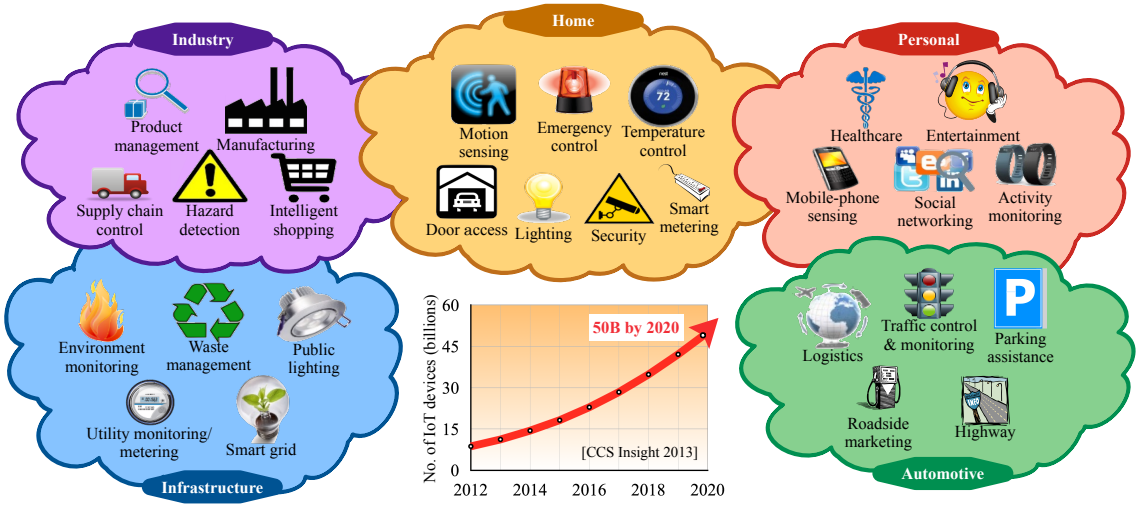


Fig. 1.1. Envisioned applications for the Internet of Things

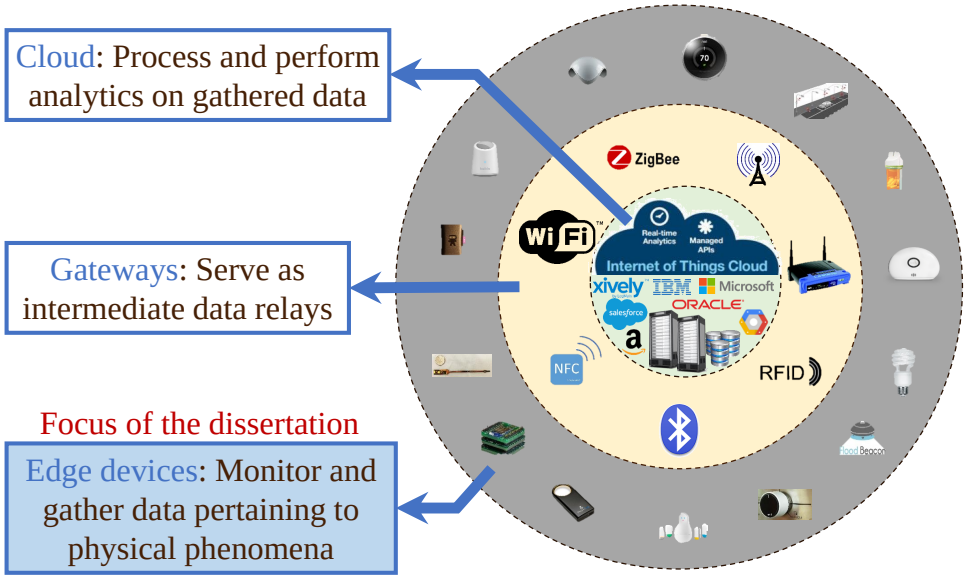


Fig. 1.2. The hierarchy of devices that make up the Internet of Things. At the center is the cloud that performs data analytics. The middle layer consists of devices that ferry the data to and from the cloud. The outermost layer consists of sensors that monitor physical phenomena. The focus of this dissertation is on the devices lying at the outermost edge of the IoT hierarchy.

layers. At the center is the computing infrastructure (cloud) that constitutes the brain of the IoT, which processes the gathered data to make inferences, learn, and take intelligent decisions. The next layer of devices includes routers, gateways, *etc.*, that form the networking infrastructure to relay the information to and from the cloud. The last layer of the IoT hierarchy consists of the devices that act as the "eyes and ears" of the IoT, sensing physical phenomenon and transmitting the gathered data to the cloud for further processing, thus bridging the physical world with the world of computing. Among the three types of devices, the last category accounts for the majority in the billions of devices that are predicted to be deployed. This dissertation focuses on these devices that constitute the outermost layer (edge) of the IoT hierarchy[1].

[1]Henceforth in this dissertation, we refer to this category of devices as IoT devices.

A major challenge in realizing the IoT vision is the problem of powering billions of devices. Most IoT devices will be wireless in the sense that they will not be powered using a cord due to the expense, inconvenience, or the sheer infeasibility of wiring them up [2, 3]. Additionally, a majority of IoT devices will be of small form-factor for a variety of reasons such as for usability, for being inconspicuous, or for adhering to space-constraints at the deployment location; limiting the space available for energy storage on these devices. Despite these constraints, many IoT devices are expected to have long operational lifetimes (from a few days to possibly several years) and work in a *set-and-forget* mode. Supplying power with batteries (as is done conventionally) is not desirable for IoT devices due to the extremely large numbers that are predicted to be in use. Each battery-powered device is associated with a maintenance cost and maintenance effort, which accounts for the labor required for replacing the battery. As the number of devices scale, the maintenance cost and effort scales in equal magnitude making frequent battery replacement not only expensive, but often infeasible. Additionally, the rapid proliferation of IoT devices will result in a surge in the number of batteries that end up in landfills, making the need to address the issue of powering IoT devices an urgent priority.

Environmental energy harvesting (where the system powers itself using energy that it harvests/scavenges from its operating environment) has long been thought of as a promising and viable option for powering these IoT devices. Energy harvesting has the advantage of eliminating maintenance overheads that accompany battery-powered systems in addition to curbing the amount of electronic waste that is generated. However, powering IoT devices with energy harvested from ambient sources (such as vibration, wind, RF signals) is challenging due to the dual constraints of deployment location and device form-factor. Deployment locations of IoT devices are always dictated by the end application, and certain deployment locations lack the availability of a copious ambient energy source that can power the device continuously. Further, any harvestable ambient source at the deployed location may be unreliable, meaning that the energy output from the source will vary with time in an unpredictable

manner. Additionally, as mentioned earlier, many of these devices are expected to be inconspicuous and compact in size, which restricts the amount of space available for having on-board energy storage that could store energy and power the device for a long time. Therefore, although powering IoT devices using energy harvested from ambient sources is a promising solution to address the power challenge, the technique suffers from drawbacks related to the often unreliable and intermittent nature of the ambient source. Combined with the inherent constraints of the IoT device, energy harvesting results in periods of power supply that are both short and intermittent and hence, pose an operational challenge for conventional IoT devices.

Intermittently-powered IoT devices are an emerging class of embedded devices that seek to utilize the energy harvested from unreliable and scanty ambient sources for their operation. Intermittently-powered systems forgo the traditional notion of stability and reliability in power supply, and operate with the knowledge that the system may lose power abruptly according to the fluctuations in ambient energy. Perceivably, performing tasks of any nature in a *reliable* manner under such power supply conditions is a daunting challenge. A fundamental requirement to make intermittently-powered systems reliable (and useful) is that any progress made during one power-on period needs to be carried forward to the subsequent periods so that the application can complete successfully (as opposed to getting stuck in a fruitless and repetitive loop of restarts and incomplete executions). Therefore, a prerequisite for making intermittently-powered systems reliable is to store (memorize) the amount of progress made during a power-on period and then later, retrieve the stored (memorized) information in the subsequent period before resuming execution. However, as we show, preexisting solutions performing the store and restore operations are *energy-inefficient* for intermittently-powered IoT devices as they adopt techniques similar to those proposed for systems of a much larger scale, and because they use Flash memory for non-volatile (persistent) storage.

The kind of non-volatile memory used in intermittently-powered systems has a significant bearing on its energy-consumption. Whenever a power loss is about to

happen, the device stores the data pertaining to the progress made by writing into the non-volatile storage. Similarly, when power is restored, the data is read back from the non-volatile storage before resuming execution. In present day IoT devices, Flash memory is used as the non-volatile memory. However, Flash memory operations are cumbersome due to the large energy and performance overhead they present. Recent advances in memory technology has seen the emergence of non-volatile memories such as Ferroelectric RAM (FeRAM), Magnetoresistive RAM (MRAM), Resistive RAM (ReRAM), *etc.*, that combine the speed, flexibility, and endurance of SRAM with the non-volatility of Flash. These emerging non-volatile memories (eNVM) are superior to Flash, both in terms of power consumption and performance, making them highly desirable choices for intermittently-powered IoT devices. This dissertation proposes system architectures that utilize the advantages of eNVM as well as SRAM to improve the energy-efficiency of intermittently-powered IoT devices.

1.1 Dissertation overview and contributions

This dissertation presents system architectures that improve the energy-efficiency of intermittently-powered systems while ensuring the reliability and forward progress of applications executing on them. We make three main contributions, namely, (a) the exploration of a unified eNVM memory architecture and *in-situ* retention scheme for intermittently-powered systems to reduce the energy overhead of storing and restoring processes; (b) the design of an energy-aware dynamic memory mapping scheme for hybrid eNVM-SRAM MCUs in intermittently-powered IoT devices to exploit both the reliability benefits of eNVM and the performance benefits of SRAM; and (c) the design of a new low power mode for deeply embedded MCUs by performing sleep mode voltage scaling to enable SRAM data retention at ultra-low power consumption. Fig. 1.3 illustrates the contributions made in this dissertation. Each contribution is summarized in some detail below.

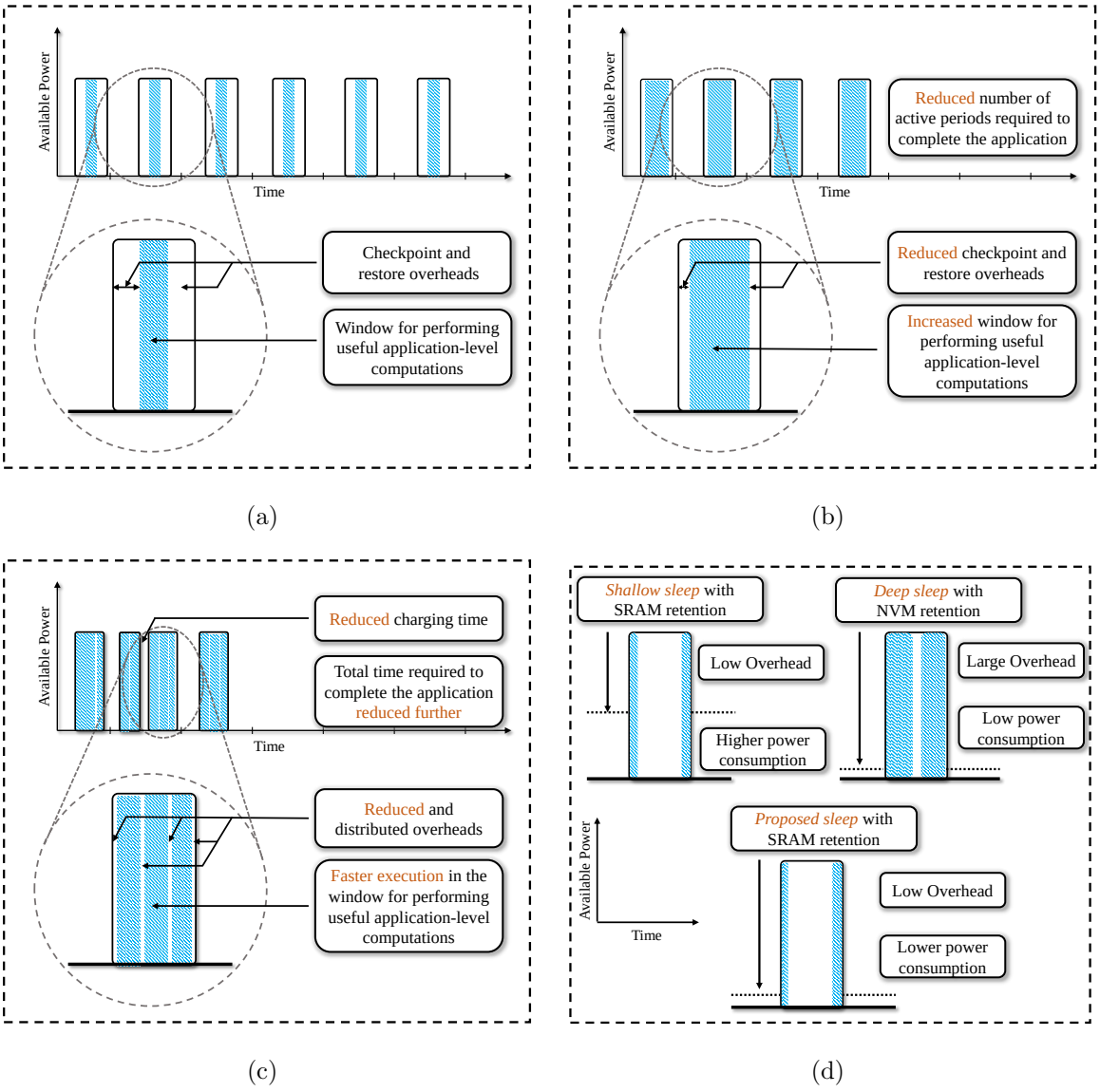


Fig. 1.3. Dissertation overview (a) Operation of typical intermittently-powered systems, wherein application executions are sandwiched between restore and checkpoint operations; (b) QuickRecall (introduced in Chapter 3) reduces checkpointing overhead; (c) Techniques presented in Chapter 4 improve overall performance by i) reducing the charging time and ii) speeding up application execution; (d) Architecture proposed in Chapter 5 introduces a new sleep mode with low overhead and low power consumption while retaining SRAM data

1.1.1 Exploration of a unified eNVM memory architecture for intermittently-powered systems

As mentioned before, intermittently-powered systems are prone to sudden and abrupt power loss due to the fluctuating nature of the ambient energy source. Therefore, it is imperative to save a snapshot of the system's state to non-volatile memory before power is lost to facilitate continuity in program execution. Conventional MCUs employ Flash as the non-volatile memory. When power loss is imminent, a snapshot (henceforth called a checkpointing operation) of system state (*e.g.,* processor registers, contents of SRAM) is stored to Flash memory, which is non-volatile. During the next burst of power, the system reboots, restores state from the stored checkpoint, and resumes program execution. Thus, long-running programs execute gradually, in small increments, as and when power becomes available. However, checkpointing to Flash involves a significant energy and time overhead due to the high erase/write power and time of Flash memory. As a result, a big portion of the time and energy when the system is `ON` is spent performing checkpointing, which limits the amount of time and energy available for program execution. Further, if the energy available in a power cycle is less than the energy required to perform a checkpoint to Flash, the IoT device can never successfully complete program execution.

Recent advances in semiconductor technology have resulted in new forms of memory technologies such as FeRAM, MRAM, ReRAM, *etc.,* that combine the speed, flexibility, and endurance of SRAM with the non-volatility of Flash, all at a very low power consumption. This has led to the possibility of *unified memory* where the same type of memory technology is used as RAM and as the non-volatile program and data storage. Low power MCUs that integrate FeRAM [4–10], MRAM [11], and ReRAM [12, 13] have already been demonstrated. This dissertation proposes QuickRecall, a hardware-software architecture that employs an emerging non-volatile memory as unified memory to enable *in-situ* checkpointing, thus alleviating the data transfer overhead for checkpoint and restore operations. QuickRecall successfully

diverts the energy that is otherwise spent on erasing and writing to Flash to perform meaningful computations, thereby improving the performance of applications in energy-harvesting IoT devices. A gist of the benefits that QuickRecall provides is illustrated in Fig. 1.3(b).

1.1.2 An energy-aware dynamic memory mapping scheme for hybrid *e*NVM-SRAM MCUs in intermittently-powered systems

For intermittently-powered IoT devices, a unified eNVM memory architecture enables *in-situ* checkpointing, thereby reducing the energy overheads required to seamlessly perform long-running computations. However, in hybrid eNVM-SRAM MCUs, a difference exists in the access latency and power consumption between the eNVM and SRAM. For example, FeRAM has a higher access latency and power consumption than SRAM. Hence, for intermittently-powered IoT devices using hybrid FeRAM-SRAM MCUs, even though a unified-FeRAM solution enables seamless computation across power cycles, it is inefficient in terms of energy consumption as compared to an entirely SRAM-based solution. On the other hand, an SRAM-based solution is highly energy efficient but unreliable as SRAM is volatile. This dissertation investigates an intermediate approach in hybrid FeRAM-SRAM MCUs that involves judicious memory mapping of program sections (`text`, `stack`, `data`, *etc.*) to retain the reliability benefits provided by FeRAM while performing almost as efficiently as an SRAM-based system, thus obtaining the best of both. Arriving at an energy-optimal memory map (where some, all, or no sections are mapped to SRAM and/or FeRAM) is challenging due to the data transfer overheads involved. Mapping sections to SRAM needs to be preceded by migration of the respective sections from FeRAM to SRAM. Similarly, on an imminent power loss, a checkpoint operation from SRAM to FeRAM needs to be performed. Therefore, the energy-optimal memory map varies from application to application depending on the memory access characteristics.

Further, ensuring reliability in intermittently-powered systems is of vital importance. This means that a successful checkpoint has to be guaranteed. However, when sections (and in particular, the `stack` section) are mapped to the SRAM, the checkpoint size becomes unpredictable. This is because the `stack` size grows and shrinks during the course of program execution making prediction of the exact checkpoint size impossible. Therefore, we propose to perform the hybrid memory mapping at the granularity of *functions*. Functions can be perceived to be independent entities that have their own `text`, `stack`, and `data` sections. Additionally, a function's memory footprint on the stack disappears after completing execution, thereby making the checkpoint size predictable and deterministic at its boundary. Hence, this dissertation performs memory-mapping to SRAM and FeRAM at the granularity of *functions* and arrives at the energy-optimal memory map. Last, we also propose a technique to proactively shut the system down in the event the remaining power in the system is insufficient to complete the execution of a function. By performing a proactive system shutdown, we reduce the charging times in addition to averting inconsistency issues that may occur due to greedy execution. Fig. 1.3(c) illustrates how mapping certain sections to SRAM might lead to faster execution of the program resulting in energy benefits.

1.1.3 Enabling SRAM data retention at ultra-low power in embedded MCUs

IoT devices that are battery-powered often work in an intermittent manner, wherein they enter the active mode for a very short duration, perform the intended task, and then enter into a low power *sleep* mode to reduce power consumption. Most MCUs provide two types of sleep modes. The first is a *shallow sleep* mode, in which the MCU core is halted, peripherals are disabled, and clock sources are turned off. However, the MCU stays powered on, which means that state information (consisting of the MCU registers and the contents of on-chip SRAM) is preserved during

sleep. Although waking up from shallow sleep is very fast, it is not the lowest power sleep mode possible. The second type of sleep mode is *deep sleep*, in which the entire MCU, including the on-chip SRAM, is powered down. While this results in the lowest power consumption possible during sleep, it does not preserve SRAM state. Hence, an energy-expensive checkpointing operation is necessary to save the state.

The reduction in power consumption of the IoT device in shallow sleep materializes from power-gating different peripheral modules and switching off sub-systems. However, the energy consumed in the sleep mode is still significant due to the large amount of time spent in the idle state. To reduce the energy consumption further, this dissertation presents Hypnos, an architecture that exploits the low retention voltage of SRAM cells to perform extreme supply voltage scaling at a system-level[2]. Hypnos implements a new ultra-low power sleep mode for MCUs that is as good as deep sleep in terms of power consumption, but still preserves the contents of SRAM, thus avoiding any data transfer overhead. The key insight behind the proposed sleep mode is the observation that the minimum voltage required for SRAM data retention is often much lower (by as much as `10x`) than the minimum operating voltage of the MCU. By lowering the supply voltage when the MCU is in sleep mode to just above the SRAM data retention voltage, Hypnos reduces the sleep mode power consumption, as shown in Fig. 1.3(d), resulting in significant energy benefits while still retaining state.

1.2 Dissertation organization

The remainder of the dissertation is organized as follows. Chapter 2 provides the necessary background for the dissertation. In particular, it lays out the background and challenges associated with intermittently-powered systems. Chapter 3 deals with the exploration of unified eNVM architecture for intermittently-powered systems and

[2]Conventionally, voltage scaling is done when the MCU is in active mode and is accompanied by a scaling of the MCU clock frequency. In contrast, here we are talking about scaling the MCU's supply voltage when it is in sleep mode.

compares the benefits of *in-situ* retention with preexisting approaches. Additionally, it provides details regarding the general hardware architecture of an intermittently-powered system and the basic modifications required to the software flow to ensure reliable forward progress of computations. Chapter 4 presents an energy-aware dynamic memory mapping scheme for hybrid eNVM-SRAM memories. It provides a comprehensive discussion on handling interrupts in intermittently-powered systems and also gives an overview on non-volatile processors. Chapter 5 presents our architecture for enabling ultra-low power SRAM retention and proposes a sleep mode, one which is as good as deep sleep while still being able to retain state. Chapter 6 concludes the dissertation and outlines possible future directions.

2. BACKGROUND

This chapter serves as the background for the rest of the chapters in this dissertation. The chapter is split into three parts and arranged as follows:

- Section 2.2 provides a brief overview on Ferroelectric RAM, the eNVM that is considered in this dissertation in some detail, which serves as the necessary background for Chapters 3 and 4.
- Section 2.3 provides a detailed background on the emerging class of IoT systems called intermittently-powered systems. In addition to providing the background, the section also provides examples of intermittently-powered systems and enlists the challenges presented by these devices with respect to energy-efficiency, reliability, and correctness. This section serves as a background for Chapters 3 and 4.
- Section 2.4 provides the background related to Chapter 5. In particular, it details the importance of addressing idle mode power consumption of microcontrollers, describes the different low power modes in present-day microcontrollers, and concludes by discussing the trade-offs associated with these different low power modes.

2.1 Introduction

Recent advances in memory technology has seen the emergence of non-volatile memories such as Ferroelectric RAM (FeRAM), Magnetoresistive RAM (MRAM), Resistive RAM (ReRAM), *etc.* These memories are random-access (like SRAM) and non-volatile like Flash. Additionally, these emerging non-volatile memories (eNVM)

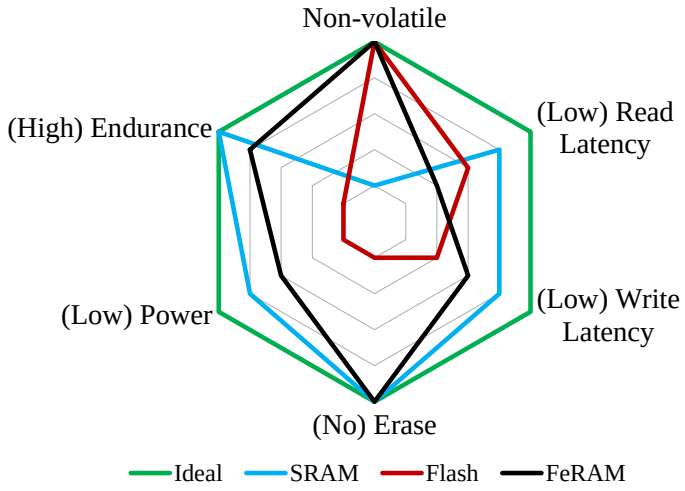


Fig. 2.1. Qualitative comparison of memory technologies

are not as energy-intensive as Flash. Inherent device limitations of Flash places constraints on how it is used. Flash memory can be written only in one-direction, *i.e.*, from 1 to 0. Additionally, data cannot be over-written to a memory location in Flash. A write operation to revert the cell from 0 to 1 requires an explicit erase operation. Further, Flash memory can be erased only at the granularity of sections (or segments), whose size depends on the Flash-memory architecture. For an MSP430 MCU that we benchmarked (in Chapter 3), the segment size was found to be 512 B. An erase operation resets all the bit-cells in the section and any data present will be lost and has to re-written again. Both the erase and write operations of Flash are energy-intensive. Comparatively, eNVM do not require an erase operation and allow data to be overwritten. This along with the random access, non-volatility, and low power characteristics make eNVM highly desirable to be used as the non-volatile memory in embedded IoT systems. Fig. 2.1 illustrates a qualitative comparison between the characteristics of various memories considered in this dissertation. In addition, the desired characteristics of a hypothetical ideal memory is also shown for comparison. FeRAM is less energy intensive as compared to Flash as it does not require an erase operation, and has lower energy consumption for writes. However, FeRAM is inferior to SRAM due to its higher access latency and power consumption. Comparatively,

SRAM is volatile, which as we will see in Chapters 4 and 5, is a cause for higher energy consumption. Since Flash and SRAM are well-known memory technologies, we divert the reader's attention instead to the characteristics of FeRAM memory technology in the following section.

2.2 Ferroelectric RAM

An FeRAM memory cell is DRAM-like in structure and uses two stable polarization states on a ferroelectric capacitor to distinguish between the dual logic levels of 0 and 1 [14–17]. The successful integration of ferroelectric memory cells with CMOS have been demonstrated from the late 1980s [18, 19] and FeRAM-integrated MCUs have since been fabricated [4–6, 8] with some of them even available commercially off the shelf [7, 9]. An FeRAM write is performed by the application of an electric field across the plates of the ferroelectric capacitor. Depending upon the direction in which the electric field is applied, the ferroelectric capacitor assumes one of the two stable polarizations corresponding to 0 and 1. An FeRAM read is accomplished by writing a particular logic value (0 or 1) to the FeRAM cell. A bit-flip or a transition to the written value requires energy, and the current pulse enabling the bit-flip is sensed to deduce the polarization of the FeRAM cell. For example, if read is done by writing a 0, and the FeRAM cell holds a logical 0, then there will be no surge in the current draw. Whereas, an FeRAM cell storing a logical 1 will flip to 0 when read, thus inducing a surge in current consumption, which is sensed. However, note that performing a read operation in this manner is destructive and requires a write-back of the read data. In present day FeRAM memories, this write-back is automatic and instantaneous, thereby presenting almost no latency overhead to the system. For the TI MSP430FR5739 MCU, which is embedded with FeRAM memory, this write-back is further protected and guaranteed against sudden power failures by using a small internal capacitor [9]. Last, the endurance of an FeRAM cell is 10^{15} cycles (which is over 10^{10} times as compared to Flash) and it retains data for over 10 years [20, 21].

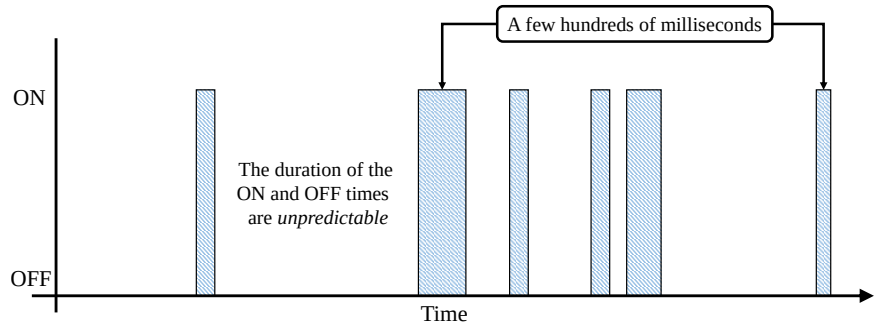
Moreover, FeRAM is random access for both reads and writes and requires no erase operation. Consequently, while Flash memory presents asymmetric read-write latencies, FeRAM access latencies are symmetric. While FeRAM as a non-volatile storage is a straightforward choice, its random access and write-in-place properties allow it to be utilized as a RAM as well, thus enabling it to serve as a unified memory technology in IoT devices.

2.3 Intermittently-powered systems: Background and challenges

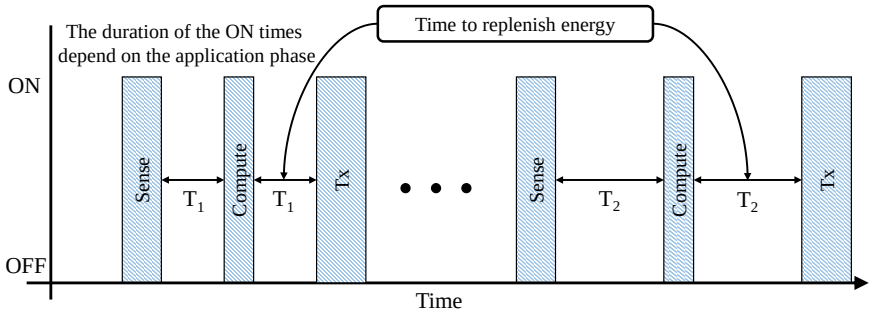
Intermittently-powered systems is an emerging class of IoT systems that are powered using the energy harvested from unreliable, unstable, and intermittent ambient sources such as that from indoor light in office corridors, RF-waves from TV and mobile-phone towers, the flow of water in a pipe, an RFID reader, *etc.* In such systems, the traditional notion of stability and reliability has to be foregone as power loss may happen in an abrupt and unpredictable manner. Therefore, executing long-running applications in a reliable manner in these systems that are subjected to frequent and sudden power loss is a daunting challenge that requires design modifications of both the embedded hardware and software architectures. In this section, we describe the scenarios in which systems may be subjected to intermittent operation and then outline the challenges in executing applications in intermittently-powered systems.

2.3.1 Intermittent operation in batteryless IoT devices

Intermittently-powered systems can be broadly classified into two categories based upon the characteristics of the ambient source from which it gathers energy and the device form-factor. The first category of systems are those that are powered by ambient sources that are both minuscule and intermittent in nature. Therefore, powering them continuously or gathering enough energy over a long period of time is challenging. Examples of such intermittently-powered systems are RFID tags (such as



(a) Intermittent operation when powered by minuscule and intermittent ambient source



(b) Intermittent operation due to form-factor, deployment-location, and ambient source constraints

Fig. 2.2. Illustration of intermittent operation in batteryless devices

the Moo CRFID [22], tags used as tollway passes, *etc.*) and contactless smart cards that are powered by RFID readers [23], WiFi-powered cameras and sensors [24], and the like. Fig. 2.2(a) illustrates the intermittent nature of operation in such devices. Whenever sufficient power is available, the system turns `ON` immediately and otherwise, remains in the `OFF` state. The duration of time that the system spends in the `ON` state depends on the availability of the ambient source at the time. Similarly, the duration of time that the system spends in the power-off state is equally unpredictable.

On the other hand, the second category consists of devices that have an ambient source available *continuously*. These devices operate intermittently as the form-factor restrictions and/or the strength of the ambient source limits the amount and rate at

which energy may be gathered. Hence, these devices are unable to operate continuously and instead utilize the energy gathered completely before losing power. Then, it spends the rest of the time in the power-off state replenishing energy. Fig. 2.2(b) illustrates the intermittent operation in these category of devices. As an example, consider a form-factor constrained device that is deployed outdoors and harvesting energy from the sunlight (similar to Senergy [25]). Due to the fact that the intensity of sunlight remains constant in shorter intervals (say over 15 minutes), the device operates in repetitive cycles. However, since the intensity of incident sunlight varies with the time of day, the duration in between successive power-on periods decrease from morning to noon and increase for the rest of the day. This fact is illustrated in Fig. 2.2(b) as the time required to replenish the system's energy (T_1 and T_2) depends on the time of day, weather, *etc.* As can be seen, $T_1 < T_2$ as the amount of sunlight received at noon is significantly larger as compared to (say) evening. Therefore, the fluctuations in the ambient energy source, which is influenced by uncontrollable factors, combined with the constraints of the device result in an intermittent power supply for these IoT systems.

Last, the system parameters such as the size of the buffer capacitor and the characteristics of the energy harvester (*e.g.*, size of the photovoltaic cell) is determined by a multitude of factors such as the device form-factor, application requirement (*e.g.*, which requires minimum n samples per hour to be collected), *etc.* This further sets limits on the minimum amount of energy that needs to be buffered per power cycle before the system turns `ON`. In this dissertation, the systems considered operate with power cycle input energies as low as 10 μJ to as much as 10 mJ[1].

[1]In most devices, the minimum energy required per power cycle is set by the communication interface. Other researches have shown the use of ambient backscattering as a low-power technique for enabling communication in these intermittently-powered systems [26, 27]. While these communication techniques work for one end of the energy spectrum, the other end that uses more conventional communication interfaces (such as Bluetooth Low Energy) sets a higher value for the minimum amount of energy per power cycle.

2.3.2 Challenges in intermittently-powered systems

The fundamental challenge in an intermittently-powered system is to ensure reliable and correct forward progress of the application executing on it even in the face of recurrent power failures. Meeting this challenge is of utmost importance as without ensuring forward progress, the application may get stuck in a never-ending loop of restarts, resets, and re-executions. Parallels may be drawn between this problem of forward progress and that of fault-tolerance in large-scale systems. In essence, to ensure forward progress, the current state of the system needs to be retained and carried forward from one powered-on cycle to the next. This method of retaining state is the adhered procedure in large-scale systems to tolerate faults and failures, such that they can resume execution from an intermediate point as opposed to restarting it from the beginning. A brief overview of the state retention techniques in large-scale systems follows.

State retention in fault-tolerant systems

Fault-tolerance or rollback recovery is a basic feature that is built into the design of large-scale systems to recover it from faults and failures occurring due to transaction aborts, hardware errors (such as memory corruption, processor failure, power outages), *etc. Checkpointing* is the technique used for tolerating premature program failures and is formally defined by the authors in Ref. [28] as *"an activity that writes information to stable storage during normal operation in order to reduce the amount of work Restart has to do after a failure"*. Checkpointing stores a snapshot of the system state into non-volatile memory (stable storage), and when a fault occurs, the system rolls back to the most recently saved checkpoint, restores it, and resumes execution [29]. Checkpointing could be done either in a transparent manner (no changes need to be made to the application program) or in a non-transparent user-directed manner [30, 31]. Transparent checkpointing entails the use of a periodic timer, whose periodic interrupts trigger the checkpoint operation. In non-transparent checkpoint-

ing, the user is handed a certain degree of control on the checkpointing overheads by being able to specify the data to be excluded (or included) in the checkpoint, and also the frequency of checkpointing by specifying locations in the code (by annotating or otherwise) for initiating the store operation. Both transparent and non-transparent checkpointing in uniprocessor systems, like the systems considered in this dissertation, cause the program to stall while the data is being written into non-volatile memory. Hence, further optimizations such as incremental checkpointing wherein only the data that has changed need to be written has also been explored [32].

Ensuring forward progress of applications in intermittently-powered systems

Checkpointing as a technique to ensure forward progress in intermittently-powered systems is not a novel concept, see Mementos [23, 33, 34]. Checkpointing, when applied in the context of an intermittently-powered system, enables the application to make progress and complete its execution across multiple power-on cycles (henceforth referred to as power cycles). Such a system would execute a part of the application in each power cycle and checkpoint the system state before an imminent power loss (at intervals). When a power loss occurs and the system wakes-up again, its state is rolled back to the last saved checkpoint and executions are resumed from that location in the program, thus making the application oblivious to the endured power loss. Giving applications this illusion of continuity is essential, without which programs could be stuck performing the same computations repeatedly in each power cycle without making any forward progress. However, a trade-off exists between ensuring forward progress of an application by checkpointing and the degradation in the application's performance due to checkpointing. To perform checkpointing, the system needs to expend precious compute cycles and energy in saving the state instead of performing relevant application-related tasks and computations. Therefore, ensuring a system's reliability (by performing checkpointing) requires a certain amount of sacrifice in its

performance. This trade-off depends on two things, namely, the checkpointing policy and rate, and the power-performance characteristics of the non-volatile memory that is employed in the system. Chapters 3 and 4 addresses this trade-off and discusses the techniques proposed in this dissertation to improve the energy-efficiency of these systems over the state-of-the-art, while executing applications in a seamless and reliable manner.

Ensuring application correctness in the face of recurrent power failures

An operational reality of intermittently-powered systems is their inability to predict the duration of time that will be spent in the involuntary powered-off state. The duration depends on the strength and availability of the ambient source and hence, is unpredictable. The time elapsed (whose length maybe unpredictable) by the system without power has implications on the validity and correctness of the checkpoint (stored state), *i.e.*, even though a checkpointing operation may be completed successfully in the previous power cycle, restoring it to resume execution in the subsequent one may not be always correct due to the time elapsed in between them. The underlying reason for this correctness issue is that some types of application state (such as previously sampled sensor data, communication parameters, *etc.*) will become stale as time advances and hence, invalid and unusable in the subsequent power cycle. Additionally, certain tasks in an application such as IO operations, non-volatile memory writes, *etc.*, cannot be split across power cycles, *i.e.*, the application cannot be resumed as is if a power loss happens while performing these tasks. In either case, if the application resumes execution, the application will either proceed in the wrong manner (as in making the wrong inferences) or get into an undefined state. Therefore, ensuring semantic correctness of the application execution is key to the proper functioning of intermittently-powered systems. Techniques that take a step in ensuring application correctness are discussed in Chapter 4 of this dissertation.

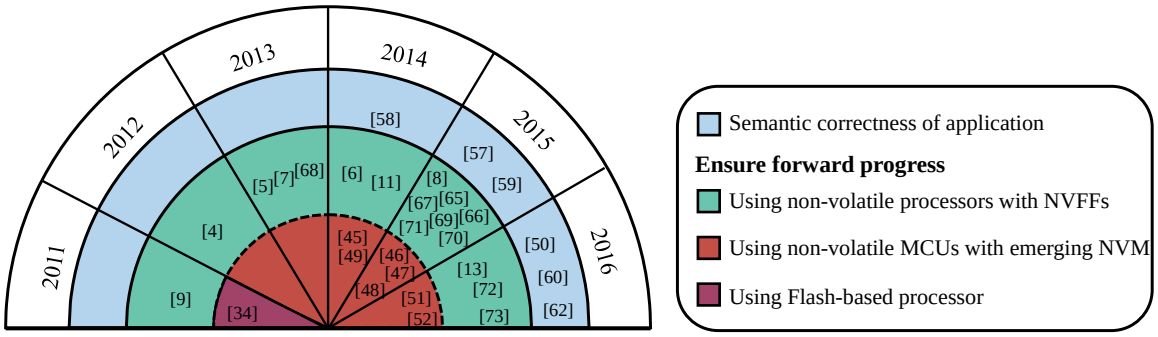


Fig. 2.3. Classification of research conducted in the field of intermittently-powered systems

Last, Fig. 2.3 shows the classification of prior work in the field of intermittently-powered systems based on the two main challenges. Related work for ensuring forward progress is discussed in Section 3.8, semantic correctness for applications in Section 4.8.2, and non-volatile processors in Section 4.8.3.

2.4 Low power modes in MCUs

Microcontrollers (MCUs) are at the heart of every embedded system that interfaces to (and interacts with) the real world. Many of the embedded systems that MCUs power, such as implantable medical devices, networked sensors, and smart meters, need to operate unattended for several years without the need for battery replacement [35, 36]. Achieving such long operational lifetime is a daunting challenge that requires extreme levels of energy efficiency. Moreover, with an estimated (>)50 billion devices predicted to be deployed in the near future [2, 3], issues such as large-scale maintenance and the resulting ecological impact compound the problem further. Fortunately, many sensing applications operate in a *heavily duty-cycled mode*, wherein the system is active only for very short bursts of time (often, only milliseconds) separated by long idle intervals (often, many tens of seconds) during which the system can be placed in a low-power, sleep mode [37, 38]. Since the sys-

tem spends greater than 99% of its time in the sleep mode, the cumulative energy spent in this mode is often the bottleneck for battery lifetime. The following quote from an ATMEL whitepaper on optimizing power consumption in MCUs sums it up well [39]: *"Although a great deal of attention is paid to active power consumption, the most important mode to consider really depends on the duty cycle between the various sleep and active modes. In applications such as thermostats, keyless entry, security systems, etc., the processor spends most of its time idle. For these applications, sleep mode may represent the lion's share of overall energy consumption and will be the most important parameter to consider."*

In order to reduce power consumption when the system is idle, a multitude of low power techniques have been proposed and implemented. A commonly-used technique is to put the microcontroller into a sleep mode that consumes lower power, and often referred to as LPM. Most MCUs provide two types of sleep modes. The first is a *shallow sleep* mode, in which the MCU core is halted, peripherals are disabled, and clock sources are turned off. However, the MCU stays powered on, which means that state information (consisting of the MCU registers and the contents of on-chip SRAM) is preserved during sleep. Although waking up from shallow sleep is very fast, it is (as expected) not the lowest power sleep mode possible. The second type of sleep mode is *deep sleep*, in which the entire MCU, including the on-chip SRAM, is powered down. While this results in the lowest power consumption possible during sleep, it does not preserve SRAM state.

Most modern microcontrollers provide multiple sleep modes, which trade-off sleep-mode power consumption with wakeup latency and data retention. Table 2.1 compares these parameters for different low power modes in a TI MSP430 microcontroller and ST-micro ARM Cortex-M0+ microcontroller. As can be seen, the MCUs provide multiple shallow LPMs that retain data. However, the disparity in power-consumption and wake-up latency between the least power-consuming shallow sleep mode and the deep sleep mode is often large. Additionally, entering and exiting a sleep mode also consumes energy and adds to the latency overhead. Therefore, this

Table 2.1.
Comparison of low power modes in present day microcontrollers

MCU	TI MSP430[a]			
Power Mode	Sleep	Standby	Off	Shutdown
Current Consumption (μA)	18	2.4	1.35	0.13
Wake-up Latency (μs)	4.5	150		2000
Data retention	Yes	Yes	Yes	No
MCU	ST ARM M0+[b]			
Power Mode	Sleep	LP Sleep	Stop	Standby
Current Consumption (μA)	1500	13	0.4	0.2
Wake-up Latency (μs)	0.25	30	51	2200
Data retention	Yes	Yes	Yes	No

[a] Values sourced from Texas Instruments MSP430F5438A MCU datasheet

[b] Values sourced from ST Micro STM32L011x4 MCU datasheet

cost needs to be also taken into account while choosing the LPM. In case of deep sleep, entering the mode without retaining data is not useful in most cases. Since all the volatile elements are powered off in deep sleep, additional energy needs to be expended to save data onto the non-volatile memory before entering deep sleep. This further adds to the energy and latency cost of entering and exiting deep sleep, which renders its use energy-inefficient for idle durations less than 40 minutes[2]. In Chapter 5, we propose a new low power mode that retains data at power consumptions lesser than that of deep sleep.

[2]Calculations for the same are shown in Chapter 5.

3. QUICKRECALL: EXPLORATION OF A UNIFIED MEMORY ARCHITECTURE FOR INTERMITTENTLY-POWERED SYSTEMS

As mentioned in Section 2, intermittently-powered systems are an emerging class of IoT devices that are powered using unreliable and often weak ambient energy sources. The unreliability in the energy source results in the power supply for the IoT device being discontinuous, which makes performing long running computations a challenging task. Examples of such systems and their operational characteristics were discussed in detail in Section 2.3.1. Fundamentally, there are two ways to address the problem of long running computations in these systems. The first one is to allocate enough energy storage (larger capacitance) such that the computation can be completed without a power interruption. While this seems a highly enticing solution to the problem, it is energy-inefficient, and not a general and scalable one. Energy-inefficiency arises due to the longer charging duration required for a larger capacitance and the fact that the energy source itself is unreliable. The leakage current of a capacitor increases with the voltage across it and therefore, spending more time to acquire the desired voltage will lead to a larger amount of useful energy being wasted. Additionally, the process of charging the capacitor may itself get interrupted causing that amount of energy to be wasted and lost as leakage. Adopting such a solution would also mean that, for the same application, each different platform would require a different value of capacitance (for collecting just enough energy). Even for the same platform, different applications or an updated version of the same application might require a different value of the capacitance. For the billion devices that are to be deployed for the IoT, such a solution is simply impractical. The second solution follows the principle of collecting energy (in small quantities) whenever it is available and then utilizing it immediately for performing computations. In conditions of extreme

unreliability in the strength and availability of the energy harvesting source, such a solution is a more viable option as it is independent of the nature of computations to be performed, platform parameters, nature of the energy harvesting source, *etc.*, and therefore, is scalable.

However, performing computations using the latter scheme requires transferring information about the progress made from one powered-on cycle to the next. Existing techniques to address this challenge are based on the idea of frequent *checkpointing*[1] of system state [34]. When power loss is imminent, a snapshot (checkpoint) of system state (*e.g.,* processor registers, contents of SRAM) is stored to Flash memory, which is non-volatile. During the next burst of power, the system reboots, restores state from the stored checkpoint, and resumes program execution. Thus, long-running programs execute gradually, in small increments, as and when power becomes available. However, checkpointing to Flash involves a significant energy and time overhead due to the high erase/write power and time of Flash memory. As a result, a big portion of the time and energy when the system is powered-on is spent performing checkpointing, which limits the amount of time and energy available for program execution. Further, if the energy available in a power cycle is less than the energy required to perform a checkpoint to Flash, the IoT device can never successfully complete program execution.

Recent advances in semiconductor technology have resulted in new forms of memory technologies such as Ferroelectric RAM (FeRAM), Magnetoresistive RAM (MRAM), *etc.,* that combine the speed, flexibility, and endurance of SRAM with the non-volatility of Flash, all at a very low power consumption[2]. This has led to the possibility of *unified memory* where the same type of memory technology is used as RAM and for non-volatile program and data storage. Low power microcontrollers with integrated FeRAM are already commercially available. For example, the TI MSP430FR5739 has 16 kB of FeRAM that can be used as unified memory [10]. This chapter makes a case

[1]Background on checkpointing is provided in Section 2.3.2.

[2]Background on emerging non-volatile memories is provided in Section 2.2.

for (and demonstrates the benefits of) using such emerging non-volatile memories in intermittently-powered systems.

3.1 Chapter contributions

In this chapter, we investigate the use of emerging non-volatile memory technologies (specifically FeRAM) in intermittently-powered systems to seamlessly enable long-running computations in the presence of frequent power interruptions. We propose a *lightweight, in-situ checkpointing technique*, called QuickRecall, for systems that use an emerging NVM. In particular, we explore a unified memory architecture wherein the same memory technology acts as both the RAM and ROM, and evaluate its impact on the energy consumption and performance of intermittently-powered systems. We show that QuickRecall can save and restore a checkpoint in just `21.06` μs, which is over two orders of magnitude lower than the corresponding overhead using Flash memory. Similarly, the energy required to save and restore a checkpoint using QuickRecall is only `30` nJ, which is over `1000x` better than a Flash-based checkpointing scheme. We have also implemented and demonstrated QuickRecall using an embedded platform called QUBE (Appendix A). Experimental evaluations using three typical embedded application programs show that the highly efficient checkpointing in QuickRecall results in a significant reduction in application-level energy consumption (as much as `3x`) and execution time (as much as `8.4x`), compared to a state-of-the-art Flash-based checkpointing technique.

3.2 Motivation for utilizing eNVM in intermittently-powered systems

As mentioned in Section 3, an intermittently-powered system turns on and performs computations whenever it receives just enough energy. The challenge posed by such systems is to perform these computations seamlessly across the powered-on cycles (henceforth referred to as power cycles) in an energy efficient manner. To successfully perform computations across power cycles, such systems require to store the

volatile system state into the non-volatile memory. By and large, microcontrollers and embedded systems of today use Flash memory as the main non-volatile storage, and SRAM as the RAM. Typically, during the process of linking a program, the linker allocates the uninitialized program sections onto the RAM for run-time initialization, whereas the global/static variables that are initialized and the program code reside on the ROM [40]. For example, in the MSP430 microcontroller, the `bss`, `data`, `heap`, and `stack` sections reside in the volatile RAM while all the other sections are allocated to the non-volatile ROM (Flash). Flash presents a significant overhead in both performance and energy due to its inherent device limitations. Flash can write (program) only from 1 to 0. A write operation to revert the cell from 0 to 1 requires it to be preceded by an erase operation. Depending on the Flash memory size and architecture, the smallest memory unit for erasure can vary. As an example, for the MSP430F5438A microcontroller, the smallest erasable unit is a segment of size `512` bytes and the erase operation takes `29` ms while consuming `320` μJ [41]. The high latency and power consumption of erase and write operations result in Flash memory having a significant energy consumption, which is a primary motivator for employing eNVM in IoT devices. In addition to being non-volatile, memories such as FeRAM and MRAM have distinct advantages over Flash in terms of power consumption, performance, endurance, *etc.* [42, 43], making it a much more suitable NVM for intermittently-powered systems than Flash memory.

3.3 Design of QuickRecall

Next, we present our proposed technique, QuickRecall, for enabling computations across power cycles in intermittently-powered systems and discuss the associated trade-offs.

3.3.1 Unified memory architecture for checkpointing

To enable computations across power cycles, the target application needs to store data pertaining to the program and processor states in a non-volatile memory before power is lost. In conventional checkpointing schemes, such checkpoint triggers are either periodic in nature or inserted at vantage locations in the program by the application designer. Checkpointing an application in this manner impedes normal program execution and introduces additional overhead.

The first element of design that QuickRecall proposes is that, for intermittently-powered systems, only a drop in the value of the supply voltage should trigger a checkpoint of the current system state. Such a checkpointing scheme does not impede normal program execution and only triggers the checkpointing if a power loss is imminent. However, one should note that in such a scheme, it is imperative that checkpointing be successfully completed before power is lost. QuickRecall ensures this by choosing an appropriate trigger voltage to interrupt the program and initiate the checkpointing operation.

The second design feature that QuickRecall introduces is to utilize a *unified memory architecture* for reducing the overhead to retain the state of an intermittently-powered system. Conventionally, the linker maps the code section to a non-volatile storage like Flash, and parts of the `data` segment such as the `bss`, `heap`, and `stack` sections to the volatile SRAM. On the contrary, QuickRecall's unified memory architecture would enable all the program sections to be mapped onto the same memory. Fig. 3.1 shows this proposed unified memory architecture along with the traditional way of mapping program sections. A unified memory architecture reduces the overheads in performing computations across power cycles. The overhead introduced comprises of checkpointing overhead and wake-up overhead. Checkpointing overhead is defined as the time required to store the system state before power is lost. Wake-up overhead is defined as the time spent in restoring the system state on power-up. The system state is made up of the states of the application program, the processor,

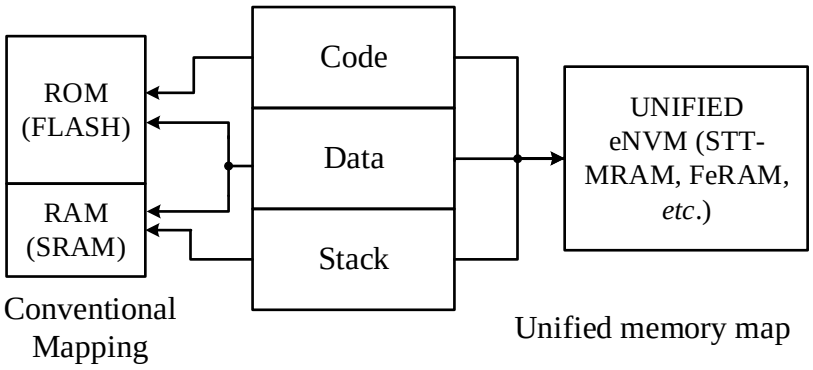


Fig. 3.1. QuickRecall's proposed unified memory architecture as compared to a conventional linking of program sections

and the state of configuration registers of various peripheral subsystems. Each of the above-mentioned state information has to be retained for a successful recall and resumption of computation across power cycles. A detailed discussion on the impact of a unified memory architecture on saving each of the system state follows.

Retaining program state

The program state consists of the values of the global variables, `stack`, `heap`, `bss`, *etc.*, in use by the program at checkpoint time. In the unified memory architecture, the eNVM acts as the conventional RAM as well as the ROM. Hence, in QuickRecall, these sections reside in the eNVM. As a result, when the MCU powers off, the RAM data is saved *in-situ*. Similarly while waking up, the program can pick up the data from exactly the same memory locations, thus avoiding the need for the checkpoint data to be rewritten to the appropriate memory locations (as is done traditionally). By using an eNVM as the RAM, QuickRecall is superior to previous checkpointing schemes as there is no time or energy overhead incurred to retain RAM data.

Retaining processor state

Processor state is defined as state of the processor at checkpoint time. To avoid re-computations after a power cycle, the processor should continue executing instructions oblivious to the incident power interruptions. Hence, capturing the processor state involves retaining the state of the microcontroller register file which includes the program counter (`PC`), stack pointer (`SP`), status register (`SR`), and General Purpose Registers (`GPRs`). The number of `GPRs` in use (alive) depends on the current state of the program. For the same program at different execution stages, variable number of `GPRs` might be in use. A software approach to track the number of active `GPRs` would hamper the normal program execution. Hence, QuickRecall saves the values of all the processor registers onto the eNVM during checkpointing. This step involves data transfer and introduces some checkpointing overhead.

Retaining microcontroller and peripheral settings

Common microcontroller applications use multiple peripherals to gather data from sensors and to communicate with the external world. The microcontroller and peripheral settings that have to be configured before execution include GPIO directions, GPIO functions, clock properties, *etc.* For intermittently-powered systems, it is pertinent to restore the MCU and peripheral states when waking up to resume correct program execution. QuickRecall addresses this problem by carefully structuring programs used for intermittently-powered systems. Depending on the application, the software boot sequence (discussed in the following section) re-initializes the configuration registers to their last known state. Note that only the registers in use by the application need to be retained. The application designer is tasked with carefully structuring the boot sequence so that the program does not enter false states. This step contributes to the wake-up overhead and the duration of the overhead is application and program dependent.

3.3.2 QuickRecall's software architecture

The QuickRecall software architecture is designed to facilitate applications to execute seamlessly across power cycles. To this end, QuickRecall's software flow introduces additional variables to the program that are hidden from the application. These extra variables are required for data retention and are allocated in the `bss` as uninitialized global variables. They include a variable to be used as the `checkpoint flag`, in addition to variables required for storing the `GPRs`, `SR`, `SP`, and `PC`. Additionally, QuickRecall's software architecture requires a dedicated GPIO to be utilized as the power-collapse warning trigger to initiate the checkpointing operation. Only QuickRecall's initialization routines are allowed to control and configure this particular GPIO, and the application is prohibited from using this GPIO. QuickRecall's software flow requires that the programmer specify the application initialization routine in a predefined function that QuickRecall's software flow uses. This restriction is placed to facilitate a successful recall of the system's peripherals upon wake-up. Last, QuickRecall modifies the boot sequence for intermittently-powered systems to ensure that the program resumes execution from the point it was paused (interrupted) before power was lost.

Fig. 3.2 shows QuickRecall's software architecture. As can be seen, the software flow has two boot sequences upon powering up. Upon boot, QuickRecall verifies the value of the `checkpoint flag` variable. An unset flag indicates a normal boot sequence, which subsequently initiates a call to the `main()` function. The `main()` function begins by initializing the MCU and peripherals, and then executes the program. While executing the application program, the MCU is interrupted on a (preselected) GPIO if the supply voltage drops below a preset trigger voltage (V_{TRIG}). Section 3.4 discusses the details and trade-offs of choosing a V_{TRIG}. Upon entering the interrupt service routine (ISR), the program context gets pushed onto the stack. QuickRecall proceeds with storing the current `SR`, `SP`, and `GPRs` in the predefined variables. Note that these registers now point to the ISR state. QuickRecall then proceeds to set the

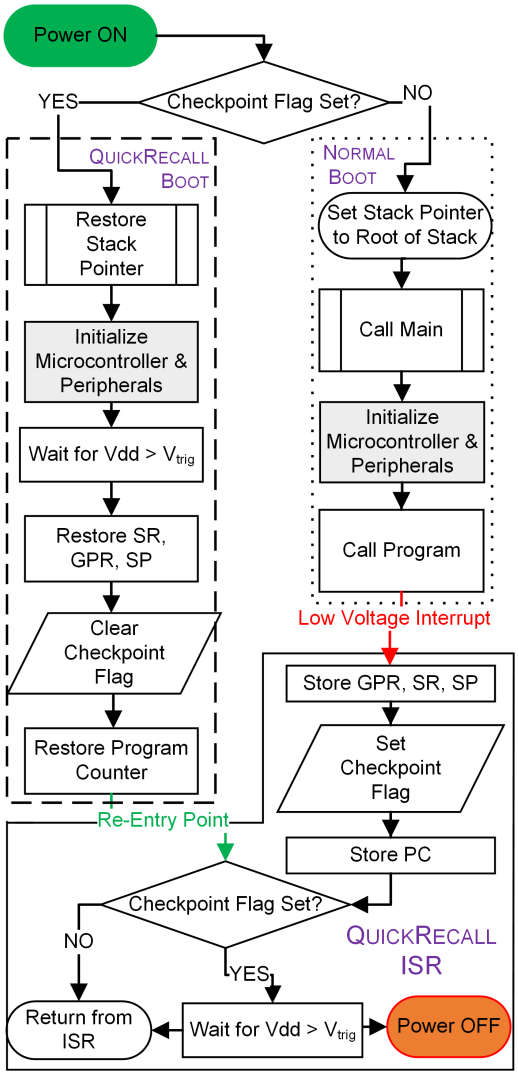


Fig. 3.2. QuickRecall's software architecture

`checkpoint flag` and saves the `PC`. Thus, the system is ready for a loss in power supply and could recall this saved state on the following boot-up event. The ISR spends any remaining time in comparing the supply voltage to the trigger voltage. If the supply voltage rises above the trigger voltage, a reverse context-switch takes place and the program continues until the supply voltage drops again. Alternatively, the microcontroller can lose power and shut off with the entire system state saved for a future recall.

On the following power up event, a set checkpoint flag launches the QuickRecall boot sequence that recalls the system state. It begins by restoring the stack pointer following which the MCU and peripheral subsystems are re-initialized. The stack pointer is restored initially so that the re-initialization routine may use the remainder of the stack without corrupting the portion of the stack corresponding to the checkpoint. QuickRecall's boot sequence then stalls execution until the supply voltage surpasses the trigger voltage. Note that, even though the peripherals have been initialized, if the MCU powers off before achieving the trigger voltage, the previous state remains intact as the ISR is not triggered. Otherwise, all the registers are reinstated and the checkpoint flag is cleared. QuickRecall then restores the `PC` and resumes by re-entering the ISR (see Fig. 3.2). The ISR returns immediately by popping the program context from the stack and the program continues execution oblivious to the power interruption.

Any application can make use of QuickRecall as long as the initialization routines are known to QuickRecall's boot sequence. QuickRecall supports all normal programming paradigms including dynamic memory allocation and nested interrupts. Dynamic memory allocation requires no additional performance overhead as the `heap` is also retained *in-situ* in the non-volatile memory. The data in the `heap` is stored as a linked-list structure in the eNVM. The memory allocation engine stores the control variables used to keep track of free and allocated `heap` segments in the `bss`. Since QuickRecall retains the state of `bss` across power cycles, the `heap` and the memory allocation engine work seamlessly across power cycles without presenting any overhead. Enabling nested interrupts facilitates the QuickRecall ISR to be triggered. Note that interrupt nesting is not enabled for the QuickRecall interrupt vector to perform checkpointing.

3.4 Hardware architecture of intermittently-powered systems

In this section, we present the details of the hardware architecture of intermittently-powered systems that are considered in this dissertation, and discuss the impact of QuickRecall on the checkpointing energy in these systems.

3.4.1 Challenges

QuickRecall's software flow is designed for a voltage-trigger based checkpointing that is more energy-efficient than the periodic monitoring of supply voltage used in other checkpointing schemes. For this purpose, an external power management unit has to be designed such that it controls the power supply to the system in addition to interrupting the system on reaching a predefined V_{TRIG}. The value of V_{TRIG} and other voltages (switch on and switch off) have to be selected in such a way as to guarantee a checkpoint. The voltages have to be carefully calibrated such that even if a system receives energy to just switch on and switch off immediately, the checkpointing operation has to be guaranteed.

3.4.2 Design

Fig. 3.3 shows the conceptual system architecture for intermittently-powered systems. The fundamental principle is to monitor the supply voltage to toggle the system between on and off states, and to trigger the checkpoint operation. The implementation of each component can vary from one system to another as long as the basic principles are adhered to.

The operation of an intermittently-powered system is governed by voltages that influences its power states. Fig. 3.4 shows the three different voltages V_{ON}, V_{TRIG}, and V_{OFF} that needs to be taken into consideration while designing an intermittently-powered system. V_{OFF} corresponds to the minimum voltage below which the MCU does not function reliably. This voltage is chosen according to the data provided by

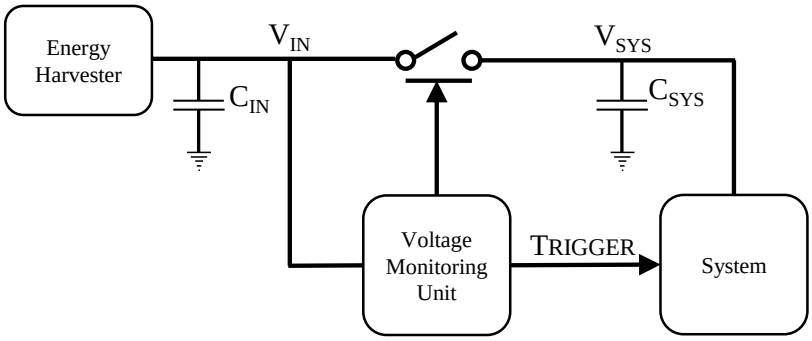


Fig. 3.3. Hardware architecture of intermittently-powered systems

the chip manufacturer. The voltage V_{TRIG} is chosen such that there is just sufficient energy to complete a successful checkpoint. The choice of V_{TRIG} depends on the V_{OFF} of the MCU and on the checkpoint energy. A conservative approach in choosing V_{TRIG} will ensure reliability but reduce energy-efficiency. This is because of the inability to predict the exact amount of energy required to complete successful checkpoint operations. As the location of the program at the time of power loss varies across power cycles, the amount of energy to checkpoint also varies from one power cycle to another. Hence, to guarantee reliability, V_{TRIG} has to be designed with the worst-case checkpoint energy. However, in the average case the energy required to complete a successful checkpoint will be much lesser than the conservative amount, which will lead to wastage and under-utilization of valuable energy. On the other hand, if the V_{TRIG} is set to a low value, it can lead to incomplete and corrupt checkpoints and affect system reliability, causing executions to be repeated and restarted from scratch. Hence, the choice of V_{TRIG} is a crucial parameter that determines the energy-efficiency of intermittently-powered systems.

The voltage V_{ON} is the voltage at which the system is designed to turn on. It has to be higher than V_{TRIG} and its minimum value depends on the checkpoint energy as well since, a larger checkpoint energy would mean a larger amount of restore energy and hence, a higher minimum V_{ON}. A hysteresis exists between V_{ON} and V_{TRIG} as the system should wake-up only if it has enough energy to perform the necessary steps

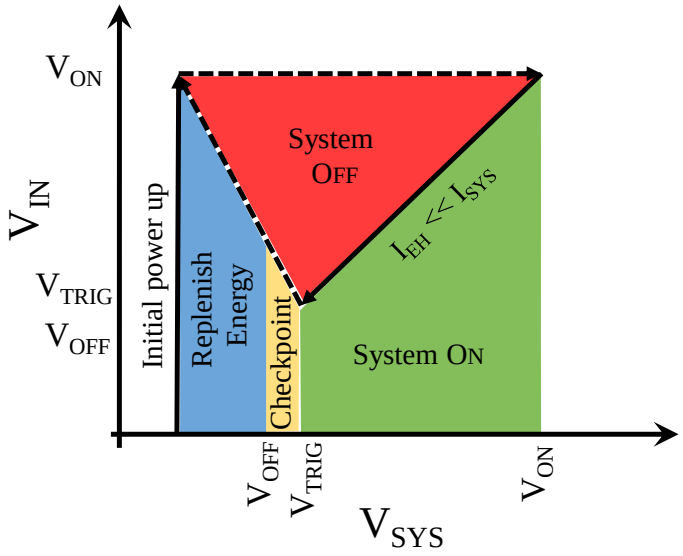


Fig. 3.4. Operational states of an intermittently-powered system in relation to energy harvester's output voltage and system voltage. The dashed lines represent a system-dependent path for the voltage transitions. 'Initial power up' corresponds to the voltage characteristics the first instance the system is powered `ON`.

after wake-up, *i.e.*, restore the system state, perform one instruction (at the very least), and successfully checkpoint the system state. Otherwise, a wake-up would lead to no meaningful usage of the gathered energy. Fig. 3.4 illustrates the system status as a function of the energy harvester output voltage (V_{EH}) and system voltage (V_{SYS}). Initially the switch (shown in Fig. 3.3) is kept open. Once V_{EH} charges to V_{ON}, the switch is closed as is depicted by the arrow. Once the system is on, the supply voltage drops until V_{TRIG} upon which the checkpoint operation is triggered. When the voltage reaches V_{OFF}, the switch is opened turning the system off and allowing the energy harvester to replenish charge. During this time, V_{SYS} may drop to a voltage ≥ 0 V and depends on the ambient energy source characteristics. In some cases, the switch is opened as early as V_{TRIG} and the checkpoint operation is performed using the energy buffered in the capacitors present in the system.

Last, the energy for checkpointing is derived from C_{SYS} as shown in Eqn. (3.1) and the energy that is input to the system at the beginning of every power cycle is given by Eqn. (3.2).

$$E_{ckpt} = \frac{1}{2} C_{SYS} (V_{TRIG}^2 - V_{OFF}^2) \tag{3.1}$$

$$E_{in} = \frac{1}{2} C_{\mathrm{IN}} (V_{\mathrm{ON}}{}^2 - V_{\mathrm{TRIG}}{}^2) \tag{3.2}$$

Note that E_{in} represents the minimum energy that is input every power cycle agnostic to the modality of (and variations in) energy harvesting.

3.4.3 Impact of QuickRecall on checkpointing energy

The voltage, V_{TRIG} is determined by the total amount of energy that is required to complete a successful checkpoint (E_{ckpt}), which is given by the equation below,

$$E_{ckpt} = E_{byte} \times N_{bytes} \tag{3.3}$$

where E_{byte} indicates the energy required for checkpointing a byte of data into NVM and N_{bytes} refers to the total number of bytes to be copied into the NVM. The value of E_{byte} depends on the kind of NVM technology in use (*e.g.*, Flash, FeRAM, MRAM, *etc.*). On the other hand, N_{bytes} depends on the program location at which the checkpoint operation is triggered, and hence varies from one checkpoint to another due to the dynamic nature of `stack` and `heap` depths, which grow and shrink during the course of program execution.

QuickRecall employs a unified memory architecture and thus, enables *in-situ* checkpointing thereby alleviating the need for copying data from the SRAM to NVM. Therefore, the amount of data that needs to be saved on the non-volatile memory is reduced to just that of the processor registers, thus reducing N_{bytes}. By adopting FeRAM as the NVM, QuickRecall reduces E_{byte} as well, thus reducing E_{ckpt} and therefore, is able to have a very low trigger voltage. Thus, QuickRecall minimizes the energy spent for checkpointing, and maximizes the energy available to be utilized for performing meaningful application tasks.

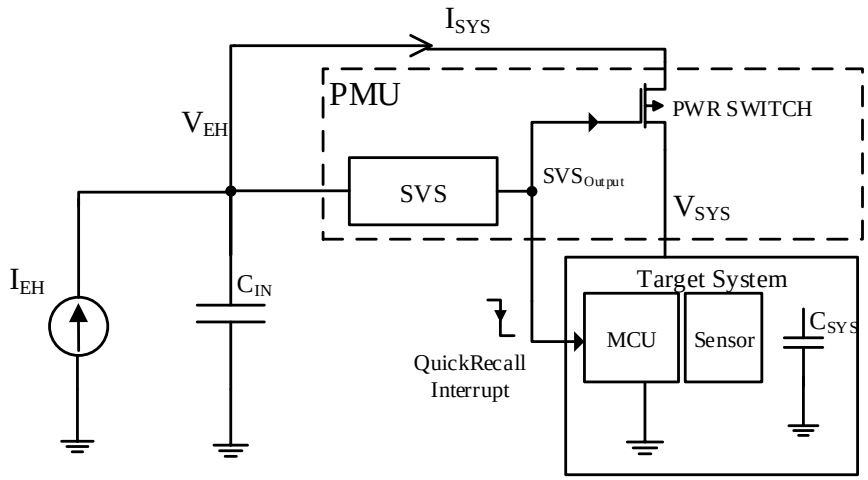
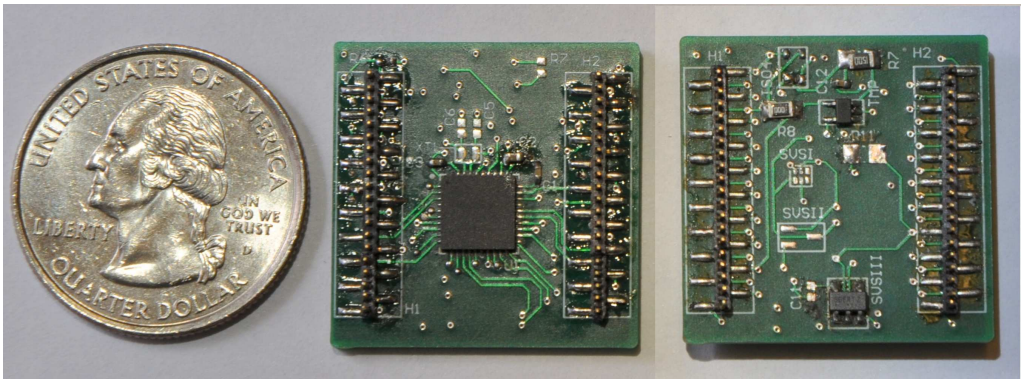


Fig. 3.5. Block diagram of QuickRecall's hardware implementation

3.5 Implementation

Fig. 3.5 shows the block diagram of our hardware implementation. In our implementation, we use FeRAM as the eNVM technology to evaluate QuickRecall. We fabricated a custom experimenter platform called QUBE[3] (Fig. 3.6) with the Texas Instruments MSP430FR5739 as the microcontroller, a temperature sensor, and a power monitoring unit (PMU). As shown in Fig. 3.5, the target system consists of the microcontroller (MCU) and the temperature sensor. The MSP430FR5739 [10] microcontroller has 1 kB of SRAM and 16 kB of FeRAM. The PMU consists of an external supply voltage supervisor (SVS) and an active-high power switch, which gates power to the target system. The SVS is interfaced with a GPIO pin of the MCU to provide a digital signal input that acts as the interrupt source for QuickRecall's software flow. Additionally, the MSP430FR5739 has a non-programmable internal SVS that monitors the microcontroller V_{DD} and regulates the voltage to the microcontroller core at a constant 1.5 V. To instrument an intermittent power supply, we use a Tektronix Keithley meter [44] as a current source along with a capacitor

[3]More details on the QUBE platform can be found in Appendix A.



(a) MCU Module (b) Sensor Module

Fig. 3.6. The QUBE platform used for evaluating QuickRecall

(C_{IN}). In all our experiments, $I_{EH} << I_{SYS}$ such that the supply capacitor (C_{IN}) takes a perceptible amount of time to charge to V_{ON}.

The choice of a suitable V_{TRIG} is crucial for QuickRecall to avoid unwanted wait periods and incomplete checkpoints. As mentioned before, the MSP430FR5739 MCU has a non-programmable internal SVS. The chosen system V_{TRIG} has to be greater than the microcontroller's internal SVS_{off} and SVS_{on} since they dictate the microcontroller on-off states. For the MSP430FR5739 MCU, the typical SVS_{off} and SVS_{on} voltages are 1.88 V and 1.93 V, respectively. The minimum voltage required for a safe FeRAM operation is 2.0 V [10]. The chosen V_{TRIG} has to guarantee correct FeRAM operation for the duration of checkpointing. The overhead of storing a checkpoint at a CPU frequency of 8 MHz, measured using an oscilloscope, is 9 μs. Therefore, we set the external SVS to a V_{TRIG} of `2.03` V with a `100` mV hysteresis using resistors. This V_{TRIG} enables QuickRecall to successfully complete checkpointing with negligible wait periods [4]. Finally, for all the experiments described in the following sections, a constant I_{EH} of 100 μA was used. The size of C_{IN} is varied to control the power cycle duration.

[4]Due to noise, the V_{TRIG} is not a strict value. A conservative V_{TRIG} which works for all power cycles was thus chosen by adjusting the decoupling capacitor size.

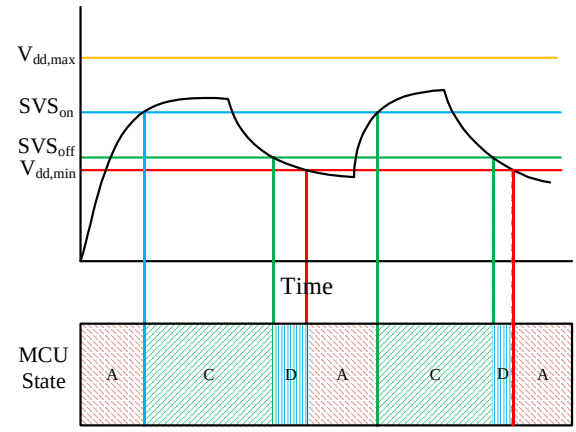
On the software front, we modified the linker to allocate the `data`, `bss`, `stack`, and `sysmem (heap)` sections to the on-chip FeRAM. Note that while the system reboots across power cycles, the global variables should not be initialized again. Hence they are defined in the `bss` section of the code. The initialization routine that configures the MCU and peripherals, like setting GPIO directions, clock frequency, *etc.*, are defined in a function (say `foo()`). `foo()` is invoked in both the `main()` function and in QuickRecall's boot sequence. Lastly, we modified the boot sequence and the initialization routines as shown in Fig. 3.2 to implement QuickRecall.

3.6 Case study: QuickRecall implementation for CRC program

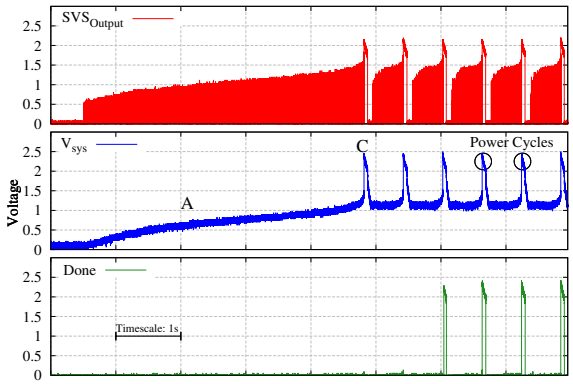
In this section, we describe how QuickRecall works for a cyclic redundancy checksum (CRC) program, when it is being executed in the face of frequent power interruptions.

Fig. 3.7(a) is a conceptual graph that shows the state of the target system (in particular the MCU) with change in V_{EH}. The (external) SVS, shown in Fig. 3.5, proctors V_{EH} and its output controls V_{SYS}. In Fig. 3.7(a), shaded region 'A' denotes the region where V_{EH} is less than the SVS_{on} voltage. The SVS keeps the power switch open, and the target system does not receive power. Once V_{EH} becomes larger than SVS_{on}, the switch is closed and V_{SYS} tracks V_{EH}. Region 'C' denotes this window when the target system receives power. As soon as V_{EH} drops below V_{TRIG}, the SVS_{Output} toggles its value, thus cutting off the power switch as well as triggering the QuickRecall interrupt. The microcontroller operation moves from region 'C' to 'D', and in 'D', the program executes the ISR to save the system state and any remaining time in this region is spent on monitoring the SVS_{Output}. The same is repeated for all subsequent power cycles. The VDD_{min} shown in Fig. 3.7(a) is the minimum voltage of operation for FeRAM in the MSP430FR5739 MCU.

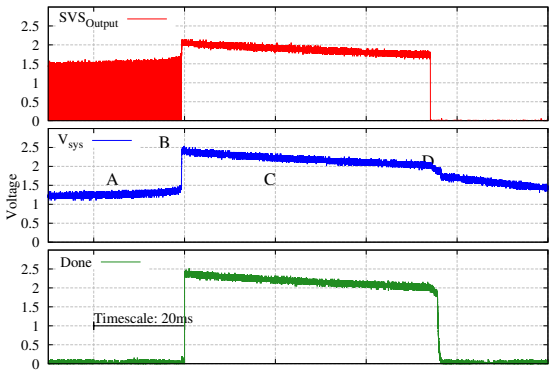
Figs. 3.7(b), 3.7(c), and 3.7(d) show real measurements of QuickRecall working across multiple power cycles for a program whose software flow is described as follows.



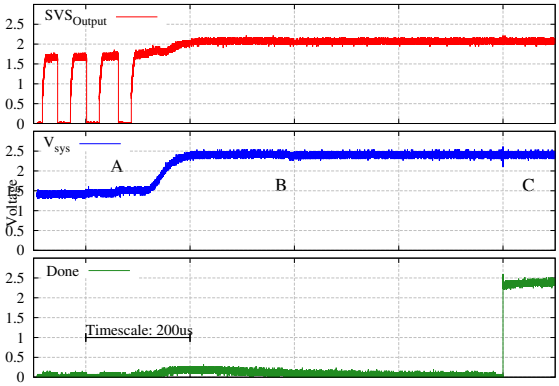
(a) V_{EH} vs. time and the corresponding MCU states



(b) Example of QuickRecall implementation for CRC



(c) Single Power Cycle



(d) System Wake up

Fig. 3.7. Detailed walk-through of MCU state transitions when employing QuickRecall in an intermittently-powered system

The program computes the cyclic redundancy checksum (CRC) of a small message. After computing the CRC, a `done` signal is raised on a GPIO of the MCU. This `done` signal is set inside a `while(1)` loop and the program persistently keeps setting it and never terminates as a result. Fig. 3.7(b) shows the SVS output, the target system voltage, and the `done` signal from the instant power is supplied. Fig. 3.7(c) is the magnified view of the fourth power cycle from Fig. 3.7(b), and Fig. 3.7(d) shows the system wake-up latency. A detailed description follows.

In the presence of an energy source, I_{EH} begins to charge C_{IN}. Once the SVS detects that V_{EH} is greater than the SVS_{on} voltage, it closes the power switch. As soon as the switch is closed, C_{IN} and C_{SYS} are connected in parallel leading to sharing of charge between the capacitors. Due to the charge sharing, V_{EH} drops below the SVS_{off} voltage (V_{TRIG}), upon which the SVS_{Output} voltage goes low, disconnecting the system from C_{SYS}. This cycle is repeated until C_{SYS} is sufficiently charged such that V_{EH} and V_{SYS} are equal to each other and more than V_{TRIG}. A magnified view of the oscillating SVS_{Output} can be seen in Fig. 3.7(d). As shown in Fig. 3.7(b), V_{SYS} shows characteristics of a typical capacitor in its charging phase. This region is marked as 'A'. During region 'A', the microcontroller's internal SVS keeps it powered off. Gradually, as C_{SYS} gets charged, the power switch remains closed for a relatively larger window and the system enters region 'B', shown in Fig. 3.7(c). In region 'B', V_{SYS} is sufficient enough to turn the MCU on, which initiates the QuickRecall boot sequence. Fig. 3.7(d) shows the transition from region 'B' to region 'C', when the MCU's previous state has been completely recalled and it commences the program execution. The MCU continues computations until it receives an interrupt from the SVS. Fig. 3.7(c) shows the transition from region 'C' to 'D'. As V_{EH} drops below V_{TRIG}, SVS_{Output} goes low, which triggers a checkpoint operation in the MCU. In region 'D', the MCU performs the said checkpoint and then waits any remaining time for the SVS output to go to logical high. In the event power fails, the microcontroller switches off with its state retained.

Fig. 3.7(b) shows that during the third such cycle, the CRC computation is complete and the `done` signal is raised. This shows that CRC computations progressed across three power cycles. In the fourth power cycle, the MCU wakes up to execute just the `while(1)` loop that constantly sets the `done` signal. This can be inferred by observing and comparing the width of the `done` signal at the third power cycle to all the ensuing ones. The first `done` signal is relatively thinner than the following ones that get set at the beginning of region 'C'. This confirms that QuickRecall works across power cycles without re-executions.

3.7 Experimental results

In this section, we first define the various terminologies used and the evaluation benchmarks, and then describe the baseline with which we present a quantitative comparison of QuickRecall.

3.7.1 Definitions

The following definitions correspond to the various parameters and metrics that we use in our experimental evaluation of QuickRecall.

Computation Window (CW)

For our experimental setup, Computation Window is defined as the time for which the MCU is in the `ON` state. This corresponds to the regions 'B','C', and 'D' as shown in Fig. 3.7(c).

Slowdown

Slowdown is defined as the ratio of time taken by the program to complete an execution across multiple power cycles to the time taken by the same code to complete executing in a single run, without any loss of power. Mathematically, if the application takes n power cycles to complete its execution, and the duration of the i^{th} power cycle is given by CW_i,

$$Slowdown = \frac{\sum_{i=1}^{n} CW_i}{TotalRuntime} \tag{3.4}$$

Slowdown happens due to the overhead presented by checkpointing schemes to store and restore the system snapshot. Note that in the above definition, the amount of time the MCU is in the OFF state does not contribute to the calculation of slowdown.

Table 3.1.
Benchmark program execution time and overhead (CPU Freq = 8 MHz)

Program	**QuickRecall Overhead** [a]	**Total Execution Time**
CRC	$12.06\mu s + 580\mu s + 9\mu s$	547ms
RSA	$12.06\mu s + 580\mu s + 9\mu s$	11.12s
SENSE	$12.06\mu s$ + 17.6ms + $9\mu s$	73ms

[a] Store Overhead + Initialization Overhead + Restore Overhead

Single Life Cycle

We define the execution of a program in a single continuous run in the absence of power loss as a single life cycle execution of the program.

3.7.2 Evaluation benchmarks

To evaluate QuickRecall, three test programs were used; namely CRC, RSA and SENSE. The CRC program calculates a 16-bit CRC and a 32-bit CRC of a message using polynomials. The RSA program does a 64-bit encryption on 128 characters. The program then decrypts the encrypted value and verifies correctness. SENSE collects temperature data from an analog sensor, processes it using a low pass filter, and then performs statistical computations such as finding the minimum, maximum, mean, and standard deviation of the collected data. SENSE implements nested interrupts as well as dynamic memory allocation on the heap [5]. The overhead introduced by QuickRecall per power cycle and the single life cycle execution runtime for each test program is given in Table 3.1

[5]Mementos, a checkpointing scheme for intermittently-powered systems, does not support dynamic memory allocation.

3.7.3 Baseline: *Flash Checkpoint*

Earlier checkpointing schemes for intermittently-powered systems are periodic in nature and make use of Flash memory as the non-volatile storage for state and data retention. For example, Mementos [34] actively polls the supply voltage at predetermined trigger points. Trigger points could be at the end of each iteration of the loop or at the end of a function call. There are some disadvantages for such a quasi-periodic checkpointing approach. Prominently, the checkpointing scheme intervenes in the normal program execution flow. This means that energy that could otherwise be utilized for performing meaningful computations is now being used to know whether the supply voltage has dropped below V_{TRIG}. Secondly, the depth of the stack size will vary depending on the program state and trigger point. This aggravates the problem of selecting a suitable V_{TRIG} since, for the checkpointing to be guaranteed in such a scheme, the chosen V_{TRIG} should satisfy a successful checkpoint of the worst case stack depth in addition to satisfying the worst case time-delay for the program to reach a trigger point. Reducing the latter implies inserting more trigger points, which in turn impedes program execution. Choosing a conservative V_{TRIG} (*i.e.* a higher value) reduces the time for meaningful computations and wastes useful energy. Hence, we reason that any checkpointing scheme that polls the supply voltage presents unwanted trade-offs as compared to an interrupt based approach.

On the other hand, QuickRecall is an interrupt based solution with no program re-executions and therefore, an unbiased baseline would employ an interrupt-based approach. When an interrupt is triggered, our baseline (henceforth referred to as *Flash Checkpoint*) initiates a data transfer of all the registers, `stack`, `heap`, *etc.*, to the Flash memory. On receiving sufficient power again, the contents from the Flash are written back to the SRAM, registers, *etc.* We define checkpoint size as the amount of data that is required to be stored in non-volatile memory at checkpoint time. Checkpoint size for *Flash Checkpoint* depends on the stack size, which grows and shrinks dynamically during the course of program execution. Since the length

Table 3.2.
Flash micro-benchmarking

Operation	**Latency**	**Avg. Power (mW)**
Read	$3.5*w + 1.9$ μs	4.55
Write	$74.8*w + 1.8$ μs	5.98
Erase	29 ms	11.05

w = no. of words

of the computation window is unknown *a priori*, no assumption can be made on the stack depth of the program at checkpoint time. Therefore, to guarantee a successful checkpointing, V_{TRIG} has to be set considering the worst case stack size of the program (*i.e.*, the worst case stack depth of CRC, RSA, and SENSE).

To measure the energy and latency overheads of Flash-based checkpointing, we use a custom MSP430F5438A board. Table 3.2 shows the characteristics of the MSP430F5438A Flash measured experimentally by performing atomic operations of read, write, and erase. The MSP430F5438A microcontroller is architecturally similar to the MSP430FR5739 MCU that is used for QuickRecall and, hence, we make use of it to make energy comparisons. Flash power measurements are done at a supply voltage of 2.6 V. One bank of the MSP430F5438A Flash is made up of 128 segments with each segment being 512 B in size. An erase operation can be done only on a single segment or on the entire bank. In Table 3.2, the read and write operation latencies are reported per number of words read/written. Using these values, we simulate a Flash-based baseline system that is described below.

Baseline Flash memory architecture and working

The Flash memory architecture that is chosen has a direct impact on the energy and latency overhead of checkpointing. Consider a Flash architecture with 2 banks

of `N` segments of size `s` words each. Assume that `n` segments (`n` < `N`) are allotted for the checkpoint operation and a checkpoint is of size `c` words. Then, a maximum of $p = \lfloor \frac{n*s}{c} \rfloor$ checkpoint operations could be performed before requiring a Flash erase operation. Typically, Flash architectures allow erase operations to be performed at a segment-level (`Segment erase`) or at a bank-level (`Mass erase`). The above architecture provides an initial benefit of p checkpoint operations without any erase. After that, either n separate segment erase operations have to be performed or a single segment erase operation have to be performed for every additional $\lfloor \frac{s}{c} \rfloor$ checkpoint operations. As Table 3.2 shows, the Flash erase operation is energy intensive and also has a latency overhead. Allotting an entire bank for checkpoint operations increases the memory footprint required and the associated cost. In case of such an architecture, an erase operation is required only once in every $\lfloor \frac{N*s}{c} \rfloor$ power cycles. Therefore, a trade-off exists between the memory footprint used and the erase overhead.

Certain Flash architectures allow the bank erase latency overhead to be hidden by concurrently permitting write access to another Flash bank. However, such an architecture still incurs an energy penalty for the erase operation albeit less frequently. The number of power cycles per erase operation is directly related to the checkpoint size of the program. The worst case checkpoint size for CRC, RSA, and SENSE are 100 B, 344 B, and 100 B respectively. Hence, *Flash Checkpoint* employs a single flash segment (of 512 B) for checkpointing operations. This segment is initially assumed to be erased and ready for checkpointing. Once the system starts computations across power cycles, checkpoint writes are performed systematically to the flash segment. The flash segment is erased only at the beginning of a computation window (CW) if it is unable to incorporate a full checkpoint for that CW. During a segment erase operation, the CPU is held and computations are stalled. Note that *Flash Checkpoint* does not benefit a huge improvement in performance by increasing the number of flash segments. This is because, most embedded system programs are non-terminating, and hence, even if more flash segments are made available for checkpointing, they will eventually be exhausted and a similar erase operation has to be performed for

each flash segment [6]. Choosing a V_{TRIG} is comparatively harder for Flash as for changes in stack depth, the energy required will vary drastically. Additionally, since Flash writes consume more energy, V_{TRIG} has to be set to a much higher value[7] than in QuickRecall to satisfy the energy requirements.

3.7.4 Quantitative comparison of QuickRecall

By reducing the checkpoint data size with *in-situ* checkpointing using FeRAM, QuickRecall reduces the latency and energy overhead for a store-restore operation, which translates into overall performance and energy benefits for the program.

As shown in Table 3.1, the QuickRecall overhead comprises of checkpoint (store) overhead and wake-up overhead. Wake-up overhead comprises of an initialization overhead and a restore overhead. Initialization overhead denotes the time spent for waking up the embedded platform, stabilizing the internal voltage regulator and PLLs, and includes the overhead for recalling the microcontroller and peripheral states. The duration of the initialization overhead is application and platform dependent. For example, SENSE has a longer wake-up overhead due to the time required for the sensor and ADC to settle. The restoring overhead is the time taken to restore the checkpoint data. For QuickRecall, this refers to the time required to populate the `GPRs`, `SR`, `SP`, and `PC` registers with their retained values upon power up. Table 3.1 also shows that QuickRecall introduces constant overheads for storing and restoring operations each power cycle. In comparison, for a Flash-based checkpointing scheme, the data has to be transferred to and from the SRAM and this overhead depends on the stack depth, number of global variables, *etc.* By employing FeRAM as a unified memory, QuickRecall implements *in-situ* checkpointing and adds zero overhead. Table 3.1 shows that the overhead related to data transfer is a constant `21.06` μs for QuickRecall. This is an improvement of `100x-1000x` over conventional checkpointing

[6]In a real deployment, increasing the number of segments may aid in wear-leveling as more segments are subjected to flash erase-write cycles

[7]Mementos uses a V_{TRIG} of `2.62`V.

Table 3.3.
Comparison of QuickRecall's energy overhead with the baseline (μJ)

Non-Volatile Memory	CRC		RSA		SENSE	
	Store	Restore	Store	Restore	Store	Restore
Flash	22.38	0.8	76.95	2.75	22.38	0.8
Flash w/ erase	342.8	-	397.4	-	342.8	-
FeRAM w/ QuickRecall	0.017	0.013	0.017	0.013	0.017	0.013

schemes using Flash. Thus, QuickRecall significantly increases the time utilized to perform meaningful computations in each power cycle.

The latency benefits that QuickRecall provides due to the reduced checkpoint size translate into reduced energy overhead for checkpointing. Equation (3.5) shows the components of energy overhead due to checkpointing and restore operations.

$$E_{power\ cycle} = P_{store} * t_{store} + P_{restore} * t_{restore} \tag{3.5}$$

In addition to reducing t_{store} and $t_{restore}$, QuickRecall reduces P_{store} and $P_{restore}$ by employing an emerging NVM instead of Flash. Note that QuickRecall's *in-situ* checkpointing scheme improves the latency overheads as compared to a case where the NVM directly replaces Flash. Further, even in a system whose Flash erase latency is hidden, the energy required for performing an erase still exists and contributes to the checkpointing overhead for Flash.

Table 3.3 quantifies and compares the energy overhead for store and restore operations for Flash and QuickRecall. The average power consumption for Flash operations are given in Table 3.2. The average power consumption measured for write and read operations of the MSP430FR5739 MCU are 1.37 mW and 1.4 mW respectively, which is atleast `4x` lower than its Flash counterpart. The energy benefits due to QuickRecall for the checkpoint operation is `1000x` better for CRC and SENSE. As can be

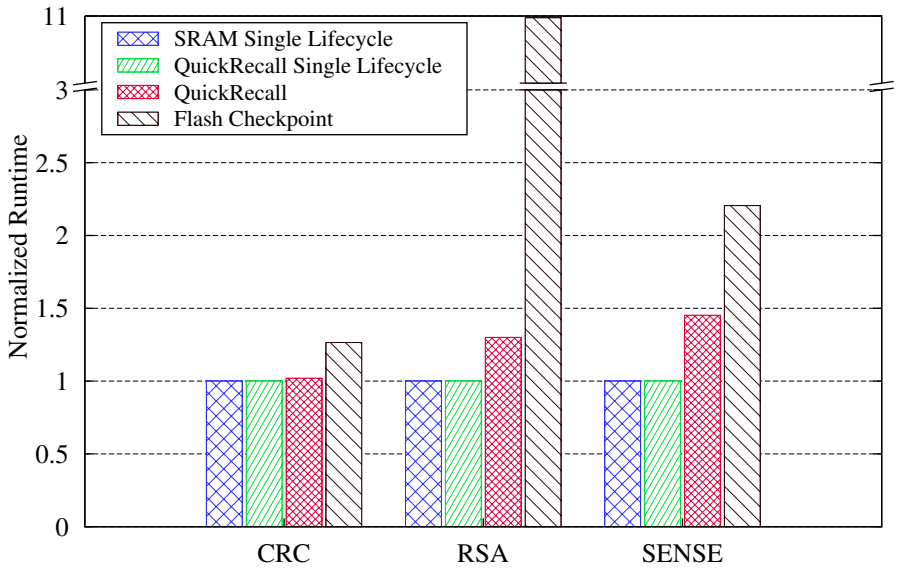


Fig. 3.8. Comparison of benchmark programs' execution times for QuickRecall and *Flash Checkpoint*

seen with RSA's store and restore energy, the Flash checkpointing energy increases as the checkpoint data size increases (by as much as `4500x` and `200x`). Additionally, the Flash energy consumption includes an intermittent erase operation that requires 320 μJ. Comparatively, QuickRecall consumes only 0.03 μJ per power cycle and is constant across programs. Even if a hypothetical microcontroller with infinite Flash memory that require no erase operations is considered, QuickRecall still reduces the energy overhead compared to Flash due to the much more compact checkpointing size that is enabled by *in-situ* checkpointing.

The energy and latency overhead reductions achieved per power cycle by QuickRecall turn into overall program-level performance and energy benefits. Fig. 3.8 compares the normalized runtime for the aforementioned benchmarks. All experiments are run with the microcontroller clock frequency set to `8` MHz. In the figure, the baseline system (normalized value of 1) is SRAM single lifecycle, in which the microcontroller system employs SRAM as the data memory. Experiments show that

the total execution time for QuickRecall single lifecycle is the same as the baseline for all the benchmarks, implying that QuickRecall does not intervene in normal program execution, and that FeRAM memory accesses are comparable to SRAM accesses at 8 MHz. For evaluating QuickRecall across power cycles, the input capacitance, C_{IN} was chosen such that the average computation window is 40 ms. The computation window for *Flash Checkpoint* simulations is also set to 40 ms. Fig. 3.8 shows that QuickRecall outperforms *Flash Checkpoint* for all three benchmarks. By utilizing FeRAM as a unified memory, QuickRecall is able to reduce the checkpoint size and data transfer overheads. CRC and RSA have the same overheads for QuickRecall and therefore theoretically, are expected to have the same slowdown. Our observation differs from the hypothesis due to the non-ideal electrical components used in a real scenario. The runtime of RSA is longer than CRC by twenty times and thus, RSA is more susceptible to variations in computation widths. The computation widths per power cycle differ slightly due to the variations in current draw from the capacitor and the MCU. Accordingly, the number of power cycles required for completion vary per experiment. For example, measurement of QuickRecall RSA with a different capacitor with an average CW of $40ms$ recorded a slowdown of 1.18x as compared to the 1.3x shown in Fig. 3.8 while the observed slowdown for CRC, being a smaller computation, remained approximately the same in both cases. On the otherhand, the slowdowns for CRC and RSA are significantly different for *Flash Checkpoint* due to the difference in stack depth for each program. SENSE, even though being the fastest of the three benchmarks, has a larger slowdown due to the aforementioned larger wake-up overhead presented during initialization at each power cycle. This consumes a significant portion of the computation window and therefore, more number of power cycles are required for SENSE to complete the computation. For *Flash Checkpoint*, more power cycles mean more checkpointing operations and correspondingly, more erase operations.

Fig. 3.9 compares the slowdown of QuickRecall with *Flash Checkpoint* when executing RSA across different computation windows. Predictably, QuickRecall does not

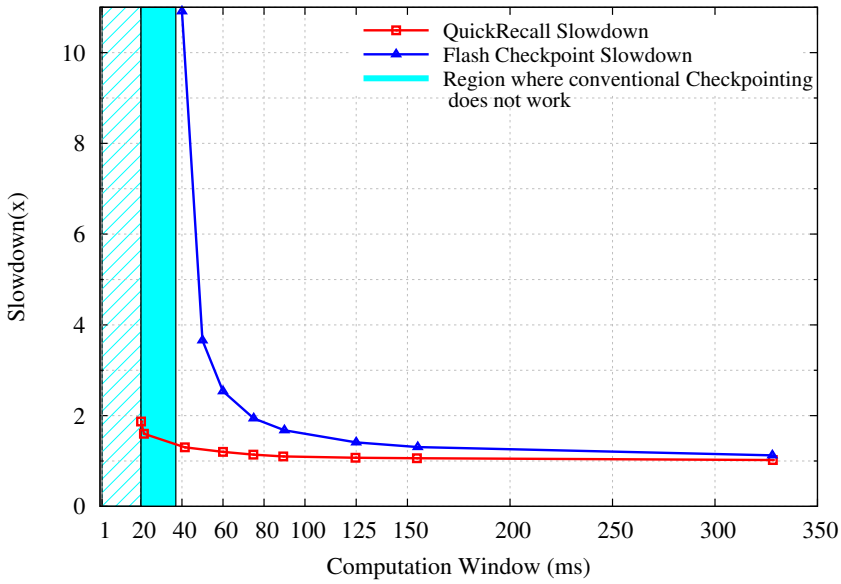


Fig. 3.9. RSA Slowdown normalized to QuickRecall Single Lifecycle

slowdown the program as much as *Flash Checkpoint* and is almost 1 for larger computation windows. Additionally, as Fig. 3.9 shows, due to the large overhead incurred for *Flash Checkpoint*, it cannot guarantee correct operation without re-executions for computation windows less than `36` ms, which is the minimum time required for an erase followed by a write operation. *Flash Checkpoint* ceases to work for computation windows less than `29` ms as erase operation cannot be performed. On the other hand, we show that QuickRecall works for computation windows as small as `20` ms without re-executions. The extremely low overhead introduced by QuickRecall gives a `1.8x` improvement in the computation window size for which the program can execute across power cycles. Theoretically, QuickRecall can do computations and successfully perform a checkpoint of the system state for computation windows as small as `1` ms. Thus, QuickRecall is a major step in enabling systems to perform computations across power cycles.

The program-level performance improvements translate into reduced energy consumption for QuickRecall. To compare the overall energy consumption of Quick-

Recall and *Flash Checkpoint*, we conducted a separate iso-input energy experiment. The QUBE platform was used for evaluating QuickRecall while a custom fabricated MSP430F5438A board was used for evaluating *Flash Checkpoint*. A capacitor of size 200 μF and a V_{ON} of 2.6 V was used for both the boards. The higher V_{ON} is chosen as it can supply the energy required for an erase operation in the case of *Flash Checkpoint*. Note that the computation windows of both the platforms are different due to the difference in computing power and energy overheads. We observed a CW of 135 ms for QUBE and 57 ms for the MSP430F5438A board. After waking up, the system restores the state and computes until the supply voltage drops to V_{TRIG}. For Flash memory, the V_{TRIG} is calculated for the worst case stack depth of a particular program as is required for a successful checkpoint. In case of RSA, the V_{TRIG} is computed to be 2.0 V. Note that for *Flash Checkpoint*, V_{OFF} is 1.8 V and not 2.0 V as is with QUBE. The platform then shuts off once the capacitor discharges to SVS_{off} voltage. One bank (64 kB) of Flash memory is allotted for checkpointing so that the frequency of erase operations is reduced. We measure the total time available for computations in a power cycle and then compute the total number of power cycles required for completing a single run of the program. Using this, the total energy consumed for QuickRecall and *Flash Checkpoint* to complete the RSA program is computed to be 22.9 mJ and 71.5 mJ respectively, *i.e.*, QuickRecall consumes 3x lesser energy as compared to *Flash Checkpoint* for executing an application across power cycles. Thus, QuickRecall enables most of the energy in a power cycle to be spent on performing meaningful computations and reduces the overall energy consumption for executing an application in an intermittently-powered system.

3.8 Related work

As mentioned in Section 2.3, checkpointing for intermittently-powered systems in not a novel concept. Ransford et al. proposed Mementos [23,33,34], which is the first work to incorporate checkpointing in the context of intermittently-powered systems.

Mementos is a compile-time technique that employs a quasi-periodic checkpointing scheme to save the state. It proposes to instrument user-written code with *trigger* points at various stages of the code such as at the end of loops and function calls. At each trigger point, Mementos polls the system voltage and performs a checkpoint operation if the voltage is below a certain predefined trigger voltage (V_{TRIG}). Since the placement of trigger points disrupts the program execution flow, Mementos also proposes a timer-based approach wherein the trigger points are enabled only in a periodic manner. Mementos keeps two checkpoints stored in its non-volatile memory and verifies the integrity of the last stored checkpoint on waking up. If the checkpoint operation was incomplete (due to the system getting powered-off midway), then Mementos picks up the checkpoint saved prior to the last one, and resumes execution from that point. However, Mementos is disadvantageous due to three reasons. First, it impedes program execution by checking the supply voltage in a proactive and repeated manner. Second, Mementos incurs a significant energy overhead by adopting Flash as the NVM. Finally, the checkpoint size varies at each trigger point, which results in Mementos requiring a large V_{TRIG} to guarantee a successful checkpoint. On the other hand, QuickRecall [45, 46] employs an interrupt-based scheme for checkpointing that triggers a checkpoint operation only once in a power cycle. By utilizing an eNVM in a unified memory architecture, QuickRecall reduces the checkpoint overhead and is able to utilize a very low V_{TRIG}.

Subsequent to QuickRecall [45], Balsamo et al. developed Hibernus [47], which utilizes FeRAM as a drop-in replacement for Flash memory as the NVM, while the SRAM is utilized as the RAM. Hibernus reduces the energy required for checkpointing by utilizing FeRAM instead of Flash, wherein the energy reduction benefits are obtained by avoiding the energy intensive erase and write operations of Flash. In comparison, QuickRecall's energy benefits spawn from using the eNVM in a unified memory architecture and thereby, reducing the checkpoint size and enabling *in-situ* retention to sidestep the data transfer latency, in addition to avoiding the energy-expensive Flash erase and write operations. Further, Hibernus differs from Quick-

Recall due to their checkpoint policy as well. In contrast to QuickRecall, Hibernus checkpoints all the peripheral registers of the MCU onto the Flash memory, thus absolving the designer from the responsibility of tracking and checkpointing the state of the registers. Such a checkpoint scheme leads to wastage of energy as the registers unused by the application will also be checkpointed in every power cycle. On the other hand, QuickRecall advocates an approach wherein the designer builds in the finite set of states that the peripherals can be in during the course of application execution, and checkpoints the particular state followed by restoring upon wake-up in the ensuing power cycle. A quantitative comparison of both the three schemes namely, Mementos, QuickRecall, and Hibernus was done by Rodriguez et al. in Ref. [48].

Other checkpoint techniques for intermittently-powered systems have since been proposed including Refs. [49–51]. The authors in Ref. [49, 51] propose incremental checkpointing schemes for intermittently-powered systems. The authors in Ref. [50] also implement an interrupt-based intermittently-powered systems, similar to QuickRecall and Hibernus. Last, a system-agnostic technique to set V_{ON} and V_{TRIG} according to the nature of the ambient energy source has since been proposed in Ref. [52].

3.9 Summary of contributions

In this chapter, we have proposed a lightweight *in-situ* checkpointing technique called QuickRecall, which enables long running computations to be executed in an energy-efficient and seamless manner in intermittently-powered systems. QuickRecall is a novel scheme that utilizes an emerging non-volatile memory in a unified memory architecture, wherein the same memory acts as both the RAM and ROM. By utilizing the unified memory architecture, QuickRecall avoids the data transfer to non-volatile memory that is otherwise required to retain the system state on an imminent power loss. We implemented QuickRecall using an MCU embedded with FeRAM and demonstrated that QuickRecall is able to perform a guaranteed checkpointing operation each power cycle by consuming only 30 nJ. Further, we showed

that QuickRecall reduces the checkpointing latency overhead by 100x-1000x over conventional Flash-based systems. We also demonstrated that the per-power cycle energy and performance benefits translate to reduction in overall application-level energy (by as much as 3x) and performance improvement (by as much as 8.4x). Thus, QuickRecall reduces the checkpoint overhead and enables most of the energy received in a power cycle to be utilized for performing meaningful computations.

4. AN ENERGY-AWARE DYNAMIC MEMORY MAPPING SCHEME FOR HYBRID eNVM-SRAM MCUs IN INTERMITTENTLY-POWERED SYSTEMS

The advent of the Internet of Things (IoT) era has fueled the emergence of new applications that improve various aspects of everyday human life. An ever-increasing number and type of IoT sensors are being deployed to seamlessly bridge the physical world with the world of computing infrastructure. However as mentioned in Chapter 1, powering such deeply-embedded IoT edge devices is extremely challenging due to their unique constraints such as remote deployment location, tiny form factor, and extreme longevity requirements. Environmental energy harvesting (where the system powers itself using energy that it scavenges from its operating environment) has been shown to be a promising and viable option for powering these IoT devices [53–55]. However, ambient energy sources (such as vibration, wind, RF signals) are often unreliable and intermittent in nature, which can lead to frequent intervals of power loss. Performing computations reliably in the face of such power supply interruptions is challenging and requires some form of *checkpointing* of system state from SRAM to non-volatile memory when power loss is imminent. Traditionally, microcontrollers have employed Flash memory as the primary non-volatile storage technology. However, the energy (and latency) intensive erase/write operations of Flash make it inefficient for frequent checkpointing.

The emergence of non-volatile memory technologies such as ferroelectric RAM (FeRAM), Resistive RAM (ReRAM), and Magnetoresistive RAM (MRAM), which have superior power and performance characteristics compared to Flash memory, has led to new hybrid memory architectures. Low power microcontrollers (MCUs) that integrate FeRAM [4, 6], ReRAM [13], and MRAM [11] have already been demonstrated. In Chapter 3, we showed that the use of FeRAM as *unified memory* (where

all program segments including `text`, `stack`, `data`, *etc.*, are mapped to the FeRAM) enables efficient *in-situ* checkpointing in IoT devices, thereby allowing them to seamlessly perform long-running computations in the face of frequent power loss. Even though FeRAM outperforms Flash in terms of performance and power consumption, it is still inferior to SRAM due to inherent device limitations. For example, in TI's MSP430FR5739 [10] microcontroller, accesses to FeRAM are `3x` slower and consume more energy as compared to SRAM. Therefore, executing programs from FeRAM results in lower performance and higher energy consumption, compared to executing programs from SRAM. On the other hand, an entirely SRAM-based solution is highly energy efficient when running continuously on reliable power, but is unreliable in the face of power loss because SRAM is volatile. This chapter advocates (and demonstrates the benefits of) an intermediate approach in hybrid FeRAM-SRAM systems that involves judicious memory mapping of program sections to retain the reliability benefits provided by FeRAM while performing almost as efficiently as an SRAM-based system, thus obtaining the best of both.

4.1 Chapter contributions

In this chapter, we investigate energy-aware memory mapping for IoT devices that are based on hybrid FeRAM-SRAM microcontrollers. We propose a comprehensive design methodology that synergistically combines the benefits of SRAM and FeRAM technologies to *efficiently, yet reliably,* perform computations across power cycles in intermittently-powered IoT systems. To that end, we propose a one-time characterization mechanism, *eM-map*, that determines the optimal memory-mapping at the granularity of functions in a program. *eM-map* performs this characterization post-deployment, which makes our solution portable. We also propose *energy-align*, a novel HW/SW technique that uses *proactive system shutdown* as a mechanism to align the time intervals when the system is powered on with function execution boundaries, which results in further improvements in energy efficiency. We implemented our

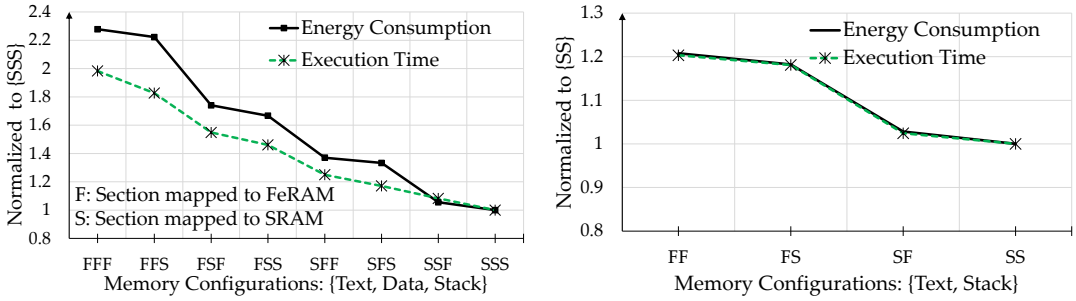
memory mapping technique on a custom hardware platform based on the Texas Instruments MSP430FR5739 MCU and have evaluated it using six typical benchmark applications used in IoT edge devices. Experimental results demonstrate performance improvements and energy savings of up to 2x and 20% respectively, compared to an existing state-of-the-art FeRAM-only solution. Last, we also discuss in detail issues pertaining to the design of intermittently power systems, particularly the validity of checkpoints to be restored, handling of interrupts, and the need for atomic execution.

4.2 Motivation for energy-aware memory mapping in intermittently-powered systems

Previous works utilizing hybrid FeRAM-SRAM MCUs for intermittently-powered systems have adopted two kinds of memory mapping schemes, namely, a unified memory mapping scheme [45, 46] and a conventional memory mapping scheme [47]. A unified memory mapping scheme maps all the program sections to FeRAM, whereas a conventional memory mapping scheme maps only the sections containing the executable code and global constants to the FeRAM while other sections are mapped onto the SRAM. Chapter 3 discusses QuickRecall, which is our proposed unified memory mapping scheme and compares QuickRecall with a conventional memory mapping scheme for an MCU that utilizes Flash as the NVM storage. For intermittently-powered systems, the memory mapping scheme has a direct bearing on its overall energy consumption. This is due to the fact that although FeRAM is better than Flash by having lesser write energy and lacking an explicit erase operation, it compares poorly to SRAM in terms of access latency. In an FeRAM-enabled MCU, MSP430FR5739 [10], we observed that the FeRAM access latency is 3x longer as compared to the on-chip SRAM. Consequently, a unified-FeRAM memory architecture will result in longer execution times. To quantify the impact that memory-mapping has on execution energy and latency, we perform an experiment wherein we explore the entire design space of possible memory maps.

For the experiment, we use the MSP430FR5739 MCU that consists of 1 kB of SRAM and 16 kB of FeRAM. Two cyclic redundancy checksum (CRC) functions are considered for evaluation and are described below. Both the functions compute the 16-bit CRC of 64 bytes of data. *CRC-I* looks up a table (of size 512 bytes) for computing the checksum and has a large memory footprint. *CRC-I* has three different sections that are of interest, namely, a `text` section that contains the executable code, a `data` section that contains the look-up table, and the `stack`. On the other hand, *CRC-II* computes CRC using polynomials and uses only the `text` and `stack` sections. For both the programs, we iteratively map each section to both FeRAM and SRAM and measure the execution time and energy consumption.

Figs. 4.1(a) and 4.1(b) show the measured energy consumption and latency associated with each memory mapping for executing the test-cases *CRC-I* and *CRC-II*. The memory mapping is represented as a 3-tuple where the first element denotes the memory map of the `text` section, followed by two elements representing the memory map of the `data` and `stack` sections. An `S` signifies that the section is mapped onto SRAM whereas an `F` indicates a mapping to the FeRAM. For example, configuration {`SFS`} in Fig. 4.1(a) signifies that the `text` and `stack` sections of *CRC-I* are mapped onto the SRAM and that the `data` section is mapped onto the FeRAM. Observe that for both the programs, a unified SRAM mapping results in the least energy consumption while a unified FeRAM mapping results in the maximum energy consumption. Overall, we note that for both *CRC-I* and *CRC-II*, any of the SRAM-mapped configurations consume less energy (by as much as `2.28x` for {`SSS`}) and execute faster (by as much as `1.98x`) as compared to the unified FeRAM configuration. However, additional data transfer operations are required for operating in a memory map configuration that has a section mapped to SRAM. This is because of two reasons. First, executing code from SRAM requires an *a priori* copy of the `text` section to the SRAM. Second, to ensure system reliability and continuity of program execution across power cycles, a checkpoint operation needs to be performed from the SRAM to the FeRAM. Hence, a trade-off exists between the data transfer cost and the execution cost for a memory



(a) *CRC-I* (b) *CRC-II* with no data section

Fig. 4.1. Energy consumption and execution time of CRC test-cases across all possible memory map configurations in a hybrid FeRAM-SRAM MCU

map configuration. Thus, an optimal memory map configuration may lie in between a unified FeRAM configuration (that has the maximum execution energy but least data transfer overhead) and a unified SRAM configuration (that has the least execution energy but maximum data transfer overhead). In this chapter, we propose a solution that finds the optimal memory map configuration, which minimizes the overall energy cost for IoT edge devices while being reliable.

4.3 Challenges in determining optimal memory map

Determining the optimal memory map configuration for a program is challenging due to the diverse nature of applications and IoT system implementations. While the diverse nature of applications make estimating the data transfer overhead challenging, the variation of system parameters from one IoT platform to another makes finding a cross-platform energy-optimal memory map infeasible. The data transfer overhead associated with executing programs from SRAM can be attributed to the processes of migration and checkpointing. Migration overhead is best defined as the energy incurred in transferring sections from FeRAM to SRAM. For example, if the considered function has the least energy consumption in configuration {`SFF`}, the executable code that resides in the non-volatile memory initially needs to be migrated

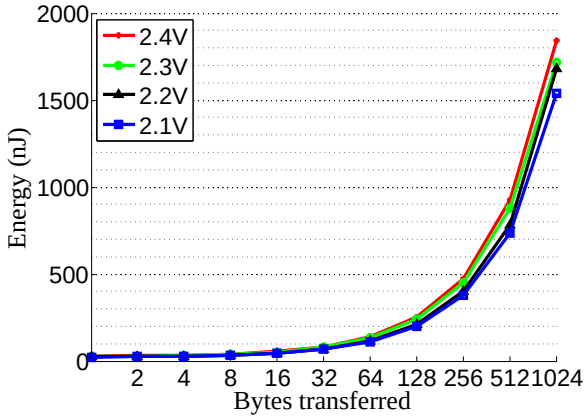


Fig. 4.2. Migration overhead

to SRAM. Migration overhead is function-dependent (*e.g.*, *CRC-II* does not use the table thus having a smaller migration overhead) and application-dependent (*e.g.*, the same function may have different input data sets when called from two locations in the program). Fig. 4.2 shows the measured energy overhead for migration as a function of the number of bytes to be migrated, for supply voltages ranging from 2.1 V to 2.4 V. From the graph, we observe that migration incurs ∼1.6 nJ per byte of data transferred from FeRAM to SRAM. Also, observe that the difference in migration overhead is negligible across the range of supply voltages used in the experiment. Therefore, migrating a section at any stage in the computing window (see definition in Section 3.7.1) in an intermittently-powered system incurs comparable energy costs.

Checkpointing, in the context of this work, is the reverse process of saving the system state from SRAM to FeRAM. Our experiments show that the energy-per-byte cost of checkpointing is similar to that of migration. However, checkpoint energy is non-deterministic due to the dynamic nature of `stack` and `heap` sizes. During the course of task execution, the stack size, heap size, *etc.*, can grow and shrink dynamically, rendering the checkpoint size and thereby the checkpoint energy unpredictable. An incomplete checkpoint results if the available energy is insufficient to save a full snapshot of the system state on an imminent power loss, leading to a loss or corruption of the system state. The energy spent in executing the program in such a

scenario is wasted and additional energy needs to be spent in program re-execution subsequently. Further, the system loses reliability in such scenarios. On the other hand, making an overly conservative estimate of the checkpoint energy will lead to under-utilization of the available energy and cause wastage. Therefore, a deterministic policy that accurately estimates the checkpoint energy among all the possible configurations is imperative in deciding the optimal memory-map.

Last, the diversity in the system parameters and device characteristics (such as sizes of on-board capacitors, power consumption of on-board components, *etc.*) of intermittently-powered systems introduce another dimension of complexity in determining the optimal memory map configuration. For example, a different value for the supply capacitor (in Fig. 3.3) could make the *CRC-I* function run to completion in a single power cycle in one IoT device but take multiple power cycles in another for the same memory-map configuration. This renders generalizing a particular memory map configuration as an optimal memory-map across all platforms impossible, affecting program portability.

4.4 Design

In this section, we first describe our design choices and then highlight the salient features of the proposed design including energy-aware memory mapping and the design of a scheme that performs proactive system shutdown.

4.4.1 Design choices for dynamic memory mapping

An important observation about the applications that are typically run on intermittently-powered systems is that they exhibit a deterministic nature in their execution flow. The typical execution flow involves an initial step wherein a physical phenomenon (*e.g.*, temperature, humidity, *etc.*) is sensed by collecting a fixed number of samples, followed by performing computations on the collected data (*e.g.*, filtering, statistical computations such as mean, standard deviation, *etc.*) that take a deterministic and

constant amount of clock cycles, and then transmitting it for further actuation or logging depending upon the energy remaining in the system. Such a relatively simple software design results in the absence of run-to-run variations in execution times, data sizes, *etc.*, thus making them predictable and deterministic.

However, the primary source of non-determinism in such systems spawns from the unpredictability in checkpoint size whenever the memory map configuration includes SRAM for data allocation. For example, when the system is memory-mapped to the {`SSS`} configuration and is about to lose power, the `stack` and `data` sections need to be copied over to the NVM. IoT applications rarely consist of self-modifying code and, therefore, the need for checkpointing the `text` section that had been migrated to SRAM is an uncommon case. Thus, the main goal of our proposed design is to reduce the non-determinism associated with the checkpoint operation, and to improve the overall performance.

Functions as the basic unit

Functions that constitute a program are by design, self-contained in terms of their sections. A function can be considered to be an independent entity having its own `text`, `data`, and `stack` sections that can be mapped onto memory at runtime. Moreover, a function also has the property that its `stack` ceases to exist upon returning to its `caller`. Therefore, performing a checkpoint at the end of a function, at its boundary, reduces the amount of data that needs to be checkpointed, which, in turn, decreases the non-determinism. Hence, we propose to perform checkpoints only at these boundaries where the checkpoint size is reduced. Fig. 4.3 illustrates our overall approach wherein each function $\texttt{foo}_\texttt{n}\texttt{()}$ in the program is an independent entity that can be mapped to either FeRAM or SRAM.

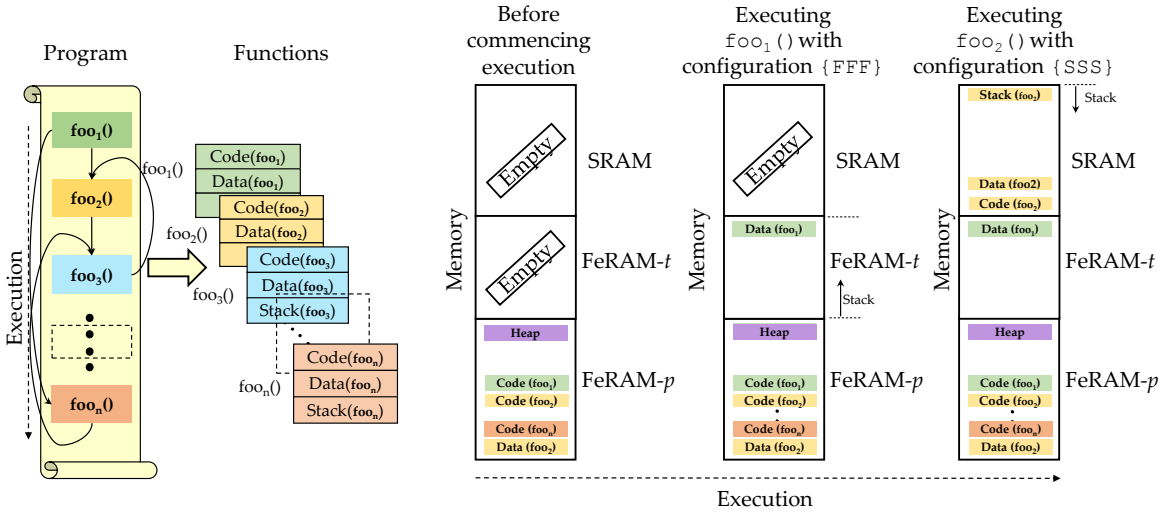


Fig. 4.3. Memory mapping for hybrid FeRAM-SRAM MCUs

Memory architecture

Fig. 4.3 also shows the memory architecture for a hybrid FeRAM-SRAM MCU. The non-volatile FeRAM memory is partitioned into two distinct regions, namely, FeRAM-p and FeRAM-t. FeRAM-p is the memory space where persistent data such as the `text` section, constants, *etc.*, are stored. FeRAM-t defines the space where a function can map temporary sections such as `stack` and `data` as dictated by the memory-map configuration during function execution, *i.e.*, FeRAM-t acts as a slower but non-volatile RAM. Note that no section is mapped to the SRAM initially. As the program executes, different functions can dynamically allocate sections onto the SRAM. Since each function is handled as an independent entity, sections that are mapped onto the SRAM become invalid once the function runs to completion. The `text` section has a fixed size and therefore occupies one end of the address space. On the other hand, the `stack` section grows and shrinks during the course of function-execution, and hence occupies the other end of the SRAM address space. The `data` section occupies the address space adjacent to the `text` section in SRAM. Note that in spite of such an arrangement, the sections may still collide during execution depending

on the total SRAM capacity and section sizes. Such memory-map configurations are inherently disallowed by our solution as explained in the following section.

Last, the `data` section inside a program can be perceived to be consisting of global variables, constants, `heap`, *etc.* In this work, we refer to the global variables, arrays, and constants as the `data` section that can be migrated between the SRAM and FeRAM. The `heap` section, which is set aside for dynamic memory allocation, is statically mapped onto the FeRAM (as shown in Fig. 4.3). The underlying reason that governs such a design decision are two-fold. First, since we propose to migrate additional sections such as `text` and `data` to SRAM, migrating the entire `heap` section will increase the chance of a collision in the SRAM and is better avoided. Additionally, copying the entire `heap` might be a futile exercise as the `heap` may not be completely utilized. Second, copying just the active part of the `heap` is challenging as it requires keeping track of the allocations made to the `heap`. The `heap` may be partially filled and fragmented, which makes tracking the active locations even more cumbersome. Hence, in this work, the `heap` is mapped exclusively to the FeRAM.

4.4.2 Energy-aware memory mapping

Arriving at the optimal memory map for a particular function requires that the energy consumption for performing the processes of migration, execution, and checkpointing be considered together. The optimal memory map for a function is one that can perform the three processes within a single power cycle with the least amount of energy. However, since the amount of input energy is dependent on the system implementation, all memory map configurations may not be possible for a function. This is because in certain memory map configurations, the energy required for migration, execution, and checkpointing may exceed the input energy per power cycle. In fact, functions may exist that cannot complete within a single power cycle for any configuration. Therefore, finding the optimal memory map configuration for a function has to be performed in an energy-aware manner. We propose *eM-map* as a one-time char-

ALGORITHM 1: *eM-map*: Energy-Aware Memory Mapping

Input : $C(F_i)$: Configuration set for each function F_i

Output: $M(F_i)$: Preferred configuration for each function F_i

Output: $E(F_i)$: Energy table for all functions

1 Pick a configuration c from $C(F_i)$;
2 $C(F_i) = C(F_i)$ - c;
3 V_{init} = measure_voltage();
4 Migrate(F_i, c);
5 Execute(F_i);
6 Checkpoint(F_i, c);
7 V_{final} = measure_voltage();
8 energy score = $f(V_{init}, V_{final})$;
9 Update_energy_table_and_preferred_config(F_i, energy score, c);
10 Shutdown();

acterization step that arrives at the optimal memory map of constituent functions of a program in an energy-aware manner. Additionally, we propose to execute *eM-map* only after deployment to ensure that the resultant memory map is energy-optimized for the particular IoT edge device, thus making *eM-map* a portable solution. A brief description of the *eM-map* algorithm (Algorithm 1) follows.

eM-map successively iterates through all possible configurations for a function to arrive at the energy-optimal configuration. The default memory map assignment is set to be configuration {`FFF`}, which corresponds to the unified FeRAM configuration. In cases wherein the function cannot complete in a single power cycle for any memory map configuration, for the sake of reliability, *eM-map* chooses the unified FeRAM configuration even though it might be not be energy optimal. Each iteration begins with the supply capacitor charged to V_{ON}. A memory map is assigned to the function and *eM-map* performs the processes of migration, execution, and checkpointing, and measures the cumulative energy consumed for all three stages. A memory map is considered valid only if the function successfully completes execution in that power

cycle. At the end of each iteration, *eM-map* updates a table with the minimum energy configuration for the considered function. A score (henceforth referred to as energy score) is calculated by function f in line 8 of Algorithm 1. This score is proportional to the energy consumption and is computed as $V_{init}^2 - V_{final}^2$, which are indicative of the initial and residual energy for the function being characterized, measured as voltages in lines 3 and 7 of the algorithm. Note that the score is independent of the capacitance, which is an invariant system parameter across iterations. Additionally, calculating the score independent of the system capacitance makes the algorithm portable across IoT devices. This score is used by *eM-map* to compare and select the energy-optimal configuration. Further, the score is also saved in the energy-table to be used for future comparisons and runtime calculations. However, if all the configurations for a function become invalid, no score can be computed and *eM-map* selects {`FFF`} as the memory configuration and denotes it in the energy table. The configuration stored in the table is then used at run-time for allocating sections to SRAM or FeRAM, while the accompanying energy score is used as a metric to govern whether a function is executable in a power cycle or not. Thus, by performing the characterization once for a device, at the granularity of functions and only a single configuration per power cycle, *eM-map* is able to find the optimal memory-map regardless of the non-deterministic nature of the data transfer overheads and agnostic to the system parameters.

4.4.3 Energy-Align: Proactive system shutdown

Energy-Align is a run-time technique that improves the energy efficiency of IoT devices that intrinsically initiates a system shutdown in an effort to reduce the charging interval in between power cycles. Algorithm 2 describes *Energy-Align* in detail. The key concept of *Energy-Align* is that it allows the execution of a function only if the system has sufficient energy to complete it. If *Energy-Align* finds that the energy remaining is insufficient, it performs a proactive shut down of the system so that it can

ALGORITHM 2: *Energy-Align*

```
Input: Energy Table: E(F_i)
Input: Memory Map Table M(F_i)
Input: F_C = Current Function
while M(F_C) is not (unified FeRAM) do
    E_rem = measure_energy_score();
    if E_rem > E(F_C) then
        Migrate(M(F_C), F_C);
        Execute(T_C);
        Checkpoint(M(F_C), F_C);
        F_C = Next Function;
    end
    else
        Shutdown();
    end
end
```

recharge until V_{ON} faster. The characterization information from *eM-map* is used to predict whether the function to be executed can be successfully completed in the current power cycle. Such an approach facilitates in reducing the energy consumption in two ways. First, *Energy-Align* ensures that migration, execution, and checkpointing of the function happens atomically. Thus, by avoiding the partial execution of functions, *Energy-Align* is able to reduce the unpredictability in checkpoint sizes, thereby avoiding energy-inefficient worst-case V_{TRIG} design. Hence, in our design, we are able to keep the trigger voltage at 2.03 V, which is the same as that of QuickRecall. By construction, *Energy-Align* will get triggered for a checkpoint at this voltage only if it runs the function in configuration {`FFF`}. For all other configurations, checkpointing happens at function boundaries and by design, the atomic operation will not extend beyond the V_{TRIG} voltage. Second, by powering the system off early, *Energy-Align*

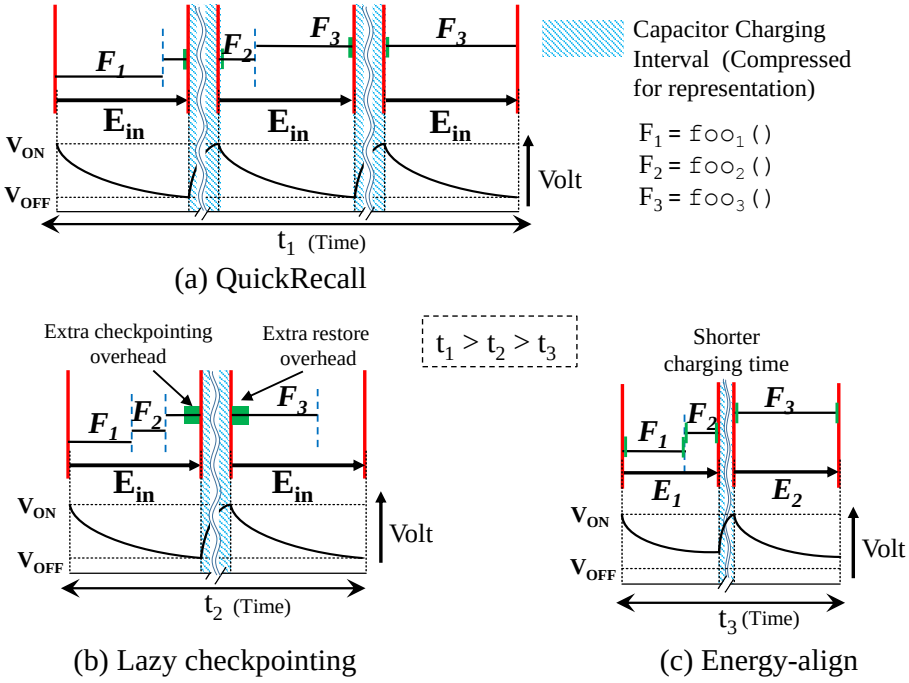


Fig. 4.4. Illustration of function-execution across power cycles for QuickRecall, *Lazy-ckpt*, and *Energy-Align*

reduces the charging time for the supply capacitor to charge back up to V_{ON}. Thus, *Energy-Align* executes the function in an energy-aware manner.

Fig. 4.4 shows the benefits of *Energy-Align* over QuickRecall and a lazy checkpointing system (henceforth referred to as *Lazy-ckpt*). The lower portion of each figure depicts the supply voltage and the top portion shows the functions F_1 through F_3 executing across power cycles. Note that the charging cycle is compressed for representation. *Lazy-ckpt* is assumed to operate in an optimal memory configuration, albeit without the capability to shut down the system to perform energy alignment. Hence, *Lazy-ckpt* has equal execution time as *Energy-Align* but incurs a significant overhead due to the conservative trigger voltage setting required for guaranteeing a successful checkpoint. As depicted in Figs. 4.4(b) and 4.4(c), *Energy-Align* and *Lazy-ckpt* run faster than QuickRecall. Additionally, note that for *Energy-Align*, functions are not split across power cycles, and hence *Energy-Align* seldom discharges the capacitor until V_{OFF}. As illustrated in Fig. 4.4(c), when the system realizes it does not have

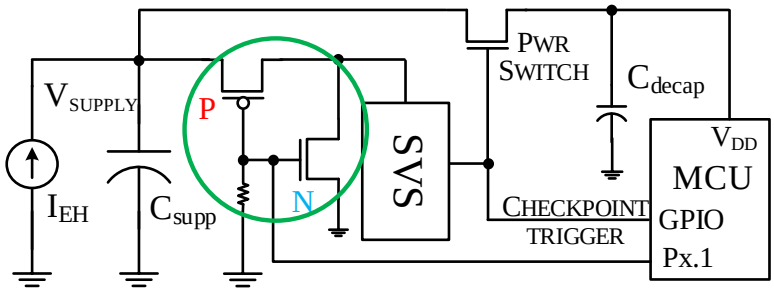


Fig. 4.5. Modified architecture for implementing *Energy-Align*

sufficient energy for executing F_3, it shuts off. Since the energy consumed by *Energy-Align* in a power cycle is lesser than the total energy available (*i.e.*, $E_1, E_2 < E_{in}$), it results in the system having a shorter capacitor charging time. Thus, *Energy-Align* improves the performance and reduces the overall energy consumption of the IoT device as compared to both *Lazy-ckpt* and QuickRecall.

Implementation of *Energy-Align*

To implement *Energy-Align*, we modified the architecture of the edge device as shown in Fig. 4.5. P connects C_{supp} to the SVS input in the default scenario. The SVS output controls the power switch that toggles the MCU between ON and OFF states. When *Energy-Align* is to be performed, the MCU pulls *Px.1* to logical high, thus momentarily connecting the SVS input to `ground` through N, and disconnects P, the path from C_{supp} to the SVS input. The path through P is disconnected so as to isolate C_{supp} from the `ground` through N and avoid unwanted discharge. Since the SVS input is grounded, power switch opens and disconnects the MCU from C_{supp}, thus switching it OFF. When the MCU turns off, the pull down resistor switches on P, thus connecting C_{supp} with the SVS input. Note that the SVS closes the power switch only when V_{SUPPLY}=V_{ON}. Since, *Energy-Align* is performed when $V_{OFF} < V_{SUPPLY} < V_{ON}$, the power switch is not closed (turned ON) immediately by the SVS. Once C_{supp} charges to V_{ON}, SVS closes the power switch, thus turning ON

the MCU. Lastly, we note that even though we utilize a commercially-available MCU that embeds FeRAM for evaluation purposes, our technique is equally applicable to MCUs with other emerging NVMs (such as MRAM, ReRAM, *etc.*).

4.4.4 Handling interrupts

In traditional embedded systems, interrupts enable the CPU to provide immediate attention to an event of high importance or are used to notify the CPU about the status of an action initiated in the past. When an interrupt is triggered, the CPU pushes its context onto the `stack` and executes the associated interrupt service routine (ISR). Typically, the execution time of ISRs are intentionally kept short and fixed so as to prevent them from taking too much time on the processor and from blocking other interrupts in the system. The time-of-arrival of an interrupt is typically influenced by factors that are either external to the system (such as notification of change in the value of the physical phenomenon being sensed, initiation of communication, *etc.*) or internal to the system (such as timers, peripherals, software initiated interrupts, *etc.*). Broadly, interrupts can be classified as deterministic and non-deterministic according to their time-of-arrival. We define deterministic interrupts as those interrupts whose time-of-arrival is expected by software. These interrupts are usually either periodic timer interrupts or notification signals indicating the completion of an action. For example, analog measurements using an ADC take a few clock cycles to converge on the value corresponding to the sensed voltage. Hence, often the arrangement between the ADC and the CPU is one wherein once the ADC conversion is initiated, it will interrupt the CPU only when the conversion completes, thus freeing the CPU to proceed with other computations or enter a low power mode. The ISR for such an interrupt is deterministic in execution time as well and typically involves copying the contents of an output buffer into a memory location. Such interrupts naturally fall within the energy analysis of *eM-map* and could be executed by *Energy-Align*. Therefore, they are not a subject of further discussion in this section.

However, non-deterministic interrupts are characterized by their unpredictability in time-of-arrival. Event-triggered interrupts fall into this class of interrupts. An example is a sensor that interrupts the system when the physical phenomenon it monitors exceeds (or falls below) a particular threshold. In systems where available energy is also a limited resource in addition to CPU time, such interrupts pose a major challenge. In particular, the ISR of a non-deterministic (and hence, unexpected) interrupt takes up energy and time away from the function that it interrupts, which may result in incomplete execution of the function in that power cycle, prompting a re-execution of the function in the next power cycle. Even though *Energy-Align* in its current form has some inherent resilience to perturbations caused by such interrupts, we enhance the robustness of our design by devising a methodology that incorporates three design elements as described below.

Design element 1 Interrupt execution supersedes function execution as with conventional interrupt design. Therefore, an interrupt will always be serviced immediately as any delay might lead to loss of state or result in a false execution. For example, an interrupt may be used to initiate data transfer from a sensor to the MCU, and any delay could cause the state to be missed or more catastrophically, a wrong state may be sensed leading to wrong inferences.

Design element 2 An interrupt service routine inherits certain characteristics from the function it interrupts such as memory mapping of the `stack`, the state of MCU peripherals, *etc.*, that influence its power consumption and execution time. Therefore, the exact energy consumed by an ISR depends on the function it interrupts and will vary from one invocation to another. For example, consider two scenarios that differ just on the kind of peripherals that are active when an interrupt triggers. The first scenario is one wherein the MCU just performs computations (and has a current consumption of 300 μA) while in the second scenario, the MCU executes a function that requires the ADC and radio peripherals to be powered on (and has a current consumption of 5.1 mA). The power consumption of the system in the latter

case will be higher due to peripherals being powered on and as a result, the ISR of an interrupt that triggers in the second scenario consumes more energy (17x) as compared to the first. Actively monitoring the number of peripheral components that are powered on at any particular stage of the program and controlling their power state for ISRs is detrimental to application execution and hence, is avoided. The other characteristic that the ISR inherits from the interrupted function is its memory map configuration, albeit partially. Since ISRs are typically short pieces of code that are executed once, the `text` section is not migrated but executed from the non-volatile memory itself. Therefore, in our design, ISRs execute with the memory map configuration of {`Fxy`} wherein `x` and `y` corresponds to the memory mapping of the `stack` and `data` sections of the function that was interrupted. Since the memory mapping influences the execution time, the function's optimal memory map affects the ISR's energy consumption as well. Hence, we choose to execute the ISR with the power and memory map configurations of the interrupted function.

Design element 3 To account for the energy required to service non-deterministic interrupts (NDIs), we allot additional energy per function as a guard-band in excess of its *eM-map* measured energy consumption. The additional amount of energy is calculated as a percentage of the function's energy consumption and is equal to $\alpha E(F_C)$, where $E(F_C)$ corresponds to the energy consumption for the function as recorded in the energy table by *eM-map* and α corresponds to a programmer-configurable fudge factor that determines the additional percentage of energy. We modify *Energy-Align* such that it compares E_{rem}, which is the energy remaining in the system, with $(1+\alpha)E(F_C)$ in line 3 of Algorithm 2 and also modify the function that updates the energy table in *eM-map* to reflect the effect of α on the memory map configuration as shown in Algorithm 3.

Since the term $\alpha E(F_C)$ corresponds to the additional energy allotted for NDIs at the beginning of each function, *Energy-Align* will determine whether the (subsequent) function is to be executed or not by comparing E_{rem} with $(1+\alpha)E(F_C)$. If E_{rem}

ALGORITHM 3: Modification for updating energy table

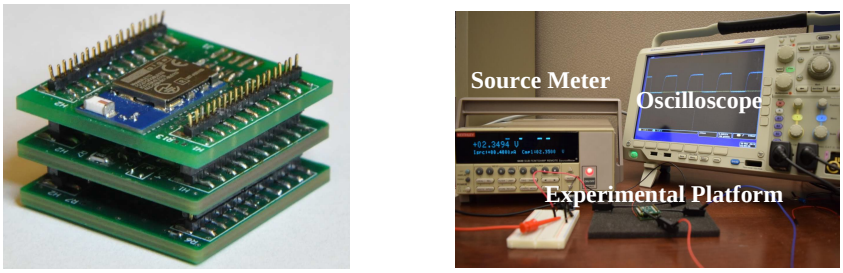
```
Input: α
Function Update_energy_table_and_preferred_config(F_i,energy,c)
    if (1 + α) × energy < E_in then
        if Update E(F_i) is success then
            Update M(F_i) with c;
        end
    end
    else
        Mark configuration as failed;
    end
```

is found to be insufficient ($E_{rem} < (1 + \alpha)E(F_C)$), then *Energy-Align* will shut-down the system and defer execution of the function to the subsequent power cycle. This ensures that the system will have $\alpha E(F_C)$ amount of energy to service NDIs in addition to that required by the function to execute successfully. The choice of value for the fudge factor α is system-dependent and therefore, is beyond the scope of this work. Although as a thumb rule, more number of NDIs in the system would mean that α should be set to a larger value. However, setting α to a value too large would impede program execution as more energy is buffered for NDIs and *Energy-Align* will defer functions with more regularity. On the contrary, setting α to a value too low reduces the number of NDIs that can be tolerated during the execution of a particular function. Hence, a small value for α increases the risk of incomplete execution of a function in the presence of multiple NDIs, which will result in re-execution of the function in the following power cycle. Last, if the function completes without using (some or all of) the energy, *Energy-Align* would automatically add the surplus amount to the next function during the process of measuring E_{rem}.

4.4.5 Discussion: Design trade-offs

As mentioned in Section 4.4.1, the applications considered in this work exhibit a deterministic nature in their execution flow. The typical execution flow of these devices involve sensing, followed by a limited set of computations before transmitting the data for further actuation or logging. Hence, they have fixed execution profiles across invocations. Extending the approach to applications with non-deterministic and dynamic execution profiles requires a modification to Algorithm 1. This is because the worst case execution time (WCET) of the function needs to be estimated before finding the optimal memory map. One approach would be to combine well-studied WCET analysis techniques [56] with *eM-map* and estimate the WCET of the function before creating the energy table. While such an approach increases application execution safety considerably, it would allot more energy than required for the average case, which will make an impact on application performance as *Energy-Align* will defer function execution with more regularity.

Another aspect of the proposed design is that the number of NDIs that are provisioned for, hinges on the system designer's ability to estimate and set α. In the case α is set low, a possibility exists wherein NDIs drain the remaining system energy, thus, leaving the function (it interrupted) with an insufficient amount of energy to complete. Preventing such a scenario requires some form of state retention prior to the power loss. Two approaches might be considered. The first one is to execute exclusively in an FeRAM only configuration (similar to QuickRecall, albeit with *Energy-Align*). However, as seen earlier (in Figs. 4.1(a) and 4.1(b)), this approach is energy-inefficient and therefore discarded. The other approach is to make a decision during the course of function execution to migrate and execute in an FeRAM-major configuration, *i.e.*, pause function execution, checkpoint and map the sections containing volatile data to FeRAM and then proceed with the remainder of the execution. However, a trade-off exists between executing in an FeRAM-major memory configuration for safety, and energy-efficiency. During the course of execution, a decision must be made by the



(a) QUBE Experimental Platform

(b) Measurement Setup

Fig. 4.6. Experimental Setup

system to checkpoint the SRAM contents (if any) to FeRAM. This operation must be triggered upon reaching a critical energy level (which we defined as the trigger voltage). As mentioned in Section 3.4.2, the choice of V_{TRIG} depends on the checkpoint energy, which depends on the amount of data that needs to be checkpointed; and as mentioned earlier, the checkpoint size is unpredictable as it depends on the program location at which the power loss happens. Hence, the V_{TRIG} would have to be set according to the worst case checkpoint size of the program in addition to having a buffer for NDIs (for safety). This conservative setting of V_{TRIG} will ensure safety by moving into an FeRAM only configuration earlier (before the power loss) at the expense of additional checkpoint overhead, higher execution cost due to FeRAM-major configuration, and performance overhead due to increased function deferring by *Energy-Align*.

4.5 Experimental results

This section describes the experimental setup, evaluation benchmarks, and the results obtained.

4.5.1 Experimental setup

Fig. 4.6 shows our experimental platform and measurement setup. The Texas Instruments MSP430FR5739 [10] microcontroller with 16 kB of FeRAM and 1 kB of SRAM is employed as the MCU. All the experiments are run with the MCU frequency set at 24 MHz. An FeRAM access takes 3 clock cycles as compared to a single cycle access for SRAM. Even though the MCU has an internal SVS, we employ an external SVS to control the power switch, and set V_{ON} and V_{TRIG}. The V_{ON} and V_{TRIG} voltages are set to 2.3 V and 2.03 V respectively. C_{IN} is 330 μF for initial experiments. A Tektronix 6430 Keithley source meter is used as the current supply, which acts as the energy harvesting module. The I_{EH} is set to 400 μA for all our experiments. Finally, all the latency overheads are recorded using a Tektronix MDO4104-3 oscilloscope.

4.5.2 Software implementation

On the software side, we implemented a modified boot-loader to incorporate *eM-map*. The boot-loader finds the V_{TRIG} of the system in the very first power cycle. The first few power cycles after deployment is spent in characterizing the function. The programmer provides the list of functions to be characterized and *eM-map* takes it as an input to create the energy table and find the optimal memory map. The default linking for the sections constituting the program is unified-FeRAM, similar to QuickRecall. To enable the boot-loader to find V_{TRIG}, QuickRecall's ISR is modified. Finally, a task manager is utilized that performs *Energy-Align* and the processes of migration, execution, and checkpointing.

4.5.3 Evaluation benchmarks

For evaluation, we consider six different applications (shown in Table 4.1) that are commonly used in IoT devices. As we mentioned in Section 4.4.1, all the ap-

Table 4.1.
Evaluation benchmarks

SnC	Sample sensor readings and perform computations
FFT	`FFT()`: Perform FFT on sampled data `Sort()`: Perform bit-reversal sorting for FFT
CRC	`CRC()`: Compute 16-bit CRC for error detection
RSA	`RSA()`: Encryption algorithm
AES	AES algorithm made up of 4 functions, namely, `addKey()`, `shiftRows()`, `mixColumns()`, and `computeKey()`
MMul	`matrixMultiply()`: Perform matrix mult. on sensor data

plications are deterministic and do not vary in their execution times or input data sizes. *Sense and Compute (SnC)* utilizes interrupts from the ADC for sampling. The interrupts are deterministic in latency and time-of-arrival, and hence cause no run-to-run variation. `FFT()` performs the fast-fourier transform on the gathered data. `Sort()` is used to perform sorting for the FFT algorithm. Other applications in the benchmark include `CRC()`, which computes the 16-bit CRC for error detection, and encryption algorithms `RSA()` and *AES*. The *AES* program consists of four functions, namely, `addKey()`, `shiftRows()`, `mixColumns()`, and `computeKey()`. Lastly, `matrixMultiply()` performs a matrix multiplication on the sensor data and stores it. Finally, we compare and evaluate our proposed solution against QuickRecall in terms of energy and latency.

4.5.4 Results

Fig. 4.7 shows the energy-rank ordering of different configurations for functions in the benchmark programs. The x-axis shows the different ranks from best to worst while the y-axis shows the possible memory-map configurations. The configurations

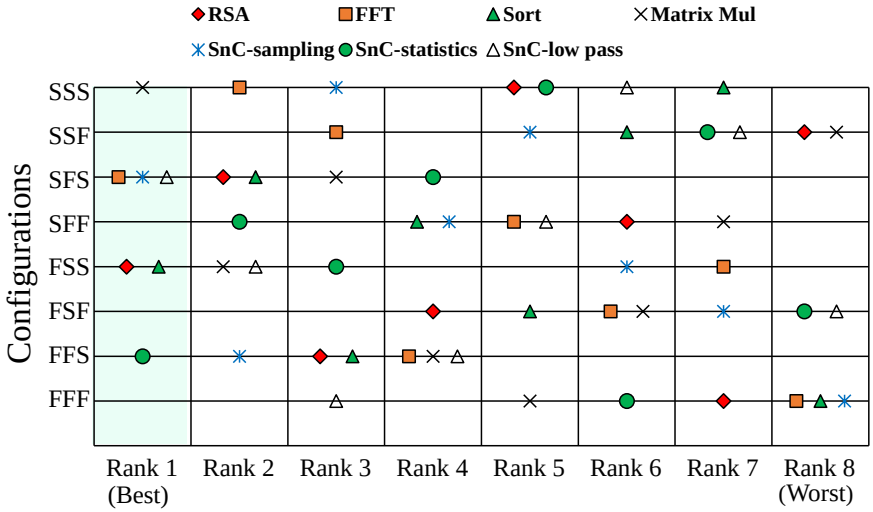


Fig. 4.7. Rank ordering of different memory map configurations

are represented in the 3-tuple format as discussed in Section 4.2. The configurations corresponding to rank 1 denote the optimal configurations for executing the function. These are output by the *eM-map* algorithm and used by *Energy-Align*. Observe that among all the different functions plotted in Fig. 4.7, only `matrixMultiply()` has the preferred configuration to be all SRAM. This means that for most functions, the data transfer overhead of migrating all the sections to SRAM is not amortized by the reduction in energy consumption achieved by executing in a unified-SRAM configuration, resulting in an optimal memory-map configuration that lies between {`FFF`} and {`SSS`}. Additionally, note that the optimal memory map configuration for all the seven functions have the `stack` section to be mapped onto the SRAM. This is due to the fact that the number of memory accesses to the `stack` is often high during the course of program execution and therefore mapping the `stack` section to SRAM has a significant impact on performance and energy consumption. For most functions, we observe that migrating the `stack` as well as just one more section of either `data` or `text` to SRAM provides the maximum energy benefits. Further, we note that the execution characteristics and memory access pattern of the function have a bearing on the observed optimal memory map configuration. This is ascer-

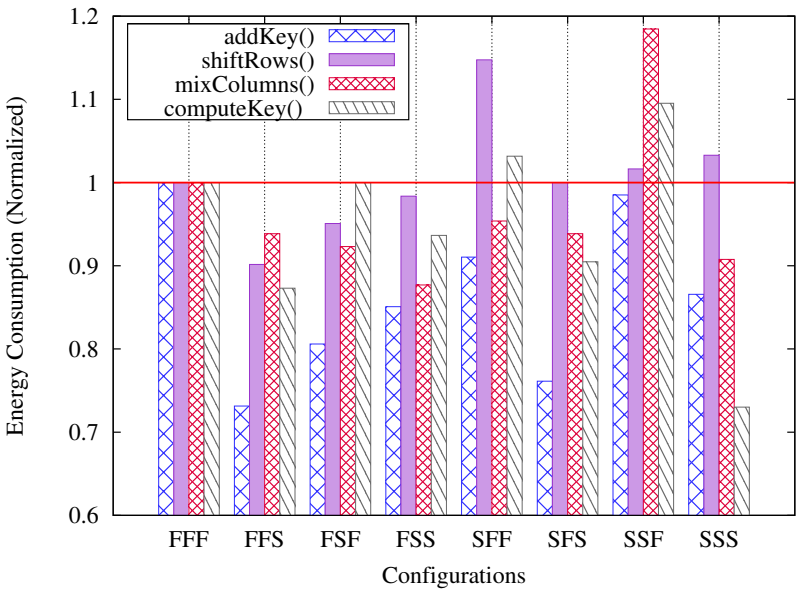


Fig. 4.8. Normalized energy consumption of different function configurations for AES

tained by the comparing the ranks of all the eight configurations of `SnC-Sampling()` and `SnC-LowPass()`. In spite of having similar sizes for `text` and `data` sections, all the ranks are different due to the fact that `SnC-LowPass()` is more stack and data intensive as compared to `SnC-Sampling()`. Note that `SnC-Sampling()` involves deterministic interrupts from the ADC, and the energy and latency overhead for the same is automatically accounted during characterization by *eM-map*.

Fig. 4.8 shows the normalized energy consumption of all the configurations for the functions in the *AES* application. The energy consumption is normalized to configuration {`FFF`}, which corresponds to QuickRecall. Note that for some functions, migration and checkpointing of sections actually result in additional energy being expended than in the {`FFF`} case. For example, even though the `shiftRows()` in configuration {`SFF`} has only the `text` section to be migrated (and nothing to be checkpointed), the overall energy consumption increases. This is because `shiftRows()` is devoid of repetitive computational kernels such as loops, and hence the cost of migration is not amortized by the reduction in execution energy. In fact, migration of the `text`

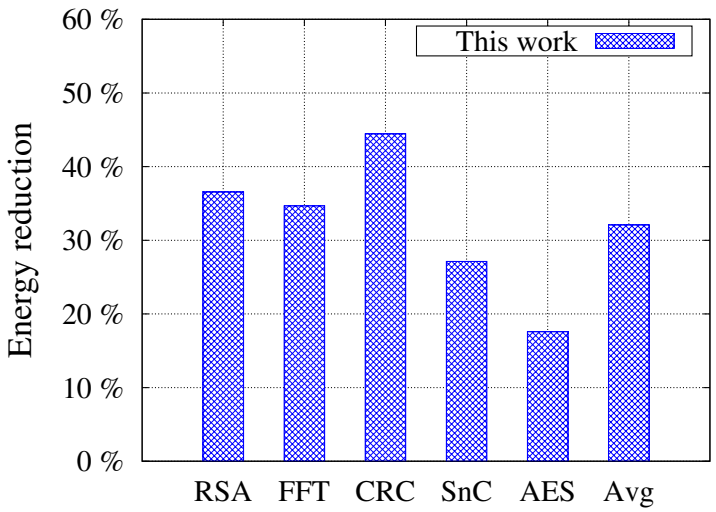


Fig. 4.9. Energy reduction in % compared to QuickRecall

section involves a read of each of the bytes in the `text` section from FeRAM, which is equivalent to executing the code once from FeRAM. Therefore, migrating the code to SRAM and then executing it is wasteful. For this reason, `shiftRows()` has the least energy benefit in its preferred configuration among the four *AES* functions. Note that the optimal configuration for all the functions have `stack` in SRAM, which concurs with our earlier observation. Overall, for the AES application, our proposed solution reduces energy consumption by `20%` as compared to QuickRecall. Fig. 4.9 shows the energy reduced consumption for each benchmark as compared to QuickRecall.

Energy measurement is an integral component in both *eM-map* and *Energy-Align*. This is achieved by a measurement of the supply voltage using the ADC that consumes ≤ 5 μJ of energy and 950 μs of latency per measurement. As Fig. 4.10 shows, this overhead is negligible as compared to the improvement in overall performance and reduction in energy consumption achieved by *Energy-Align*. Fig. 4.10 shows the execution times of different IoT applications normalized to QuickRecall for a single run of the application across power cycles. The execution time includes the time required by the capacitor to regain charge and switch-on the system. As is evident, the energy-aware memory-mapped solution has better performance (as much as `2x`)

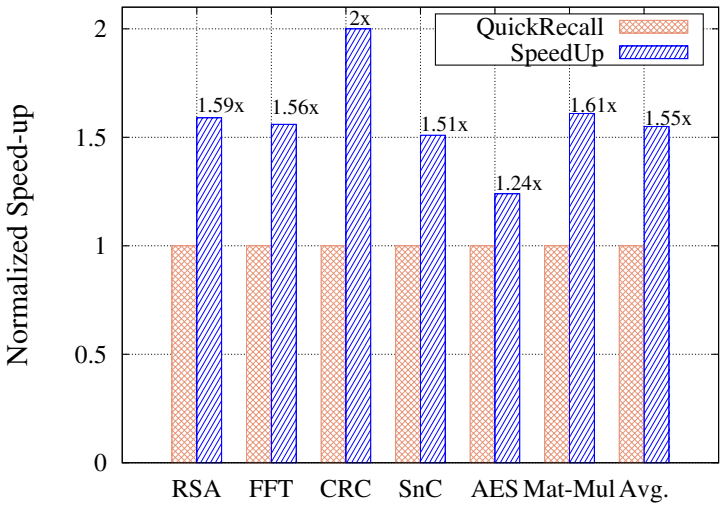


Fig. 4.10. Speed-up comparison normalized to QuickRecall

Table 4.2.
Rank order of configurations for the FFT-Sort benchmark using two different C_{IN} (N.V.= not valid)

C_{supply}	FFF	FFS	FSF	FSS	SFF	SFS	SSF	SSS
330 μF	8	3	5	1	4	2	6	7
180 μF	4	3	N.V.	1	N.V.	2	N.V.	N.V.

as compared to QuickRecall. The speed-up stems from the reduction in execution time achieved by energy-efficient memory mapping of sections by *eM-map* and also from the reduction in charging time achieved by *Energy-Align*. Note that, even if two applications have the same overall migration overhead and optimal configurations, the unique characteristics of function-execution and memory access patterns result in different speed-ups.

Last, to show that *eM-map* is agnostic to system parameters, we run the algorithm again for the `FFT-Sort()` function with C_{IN} set to 180 μF. Results of the experiment are shown in Table 4.2. Most of the configurations fail to execute successfully in a

single power cycle in the new system rendering them invalid (shown as N.V.). *eM-map* assigns the last outstanding rank to {FFF}, which is marked as rank 4 in the table. We also note that the memory-map configuration output by *eM-map* is agnostic to any variations in the input power trace. This is due to two reasons. First, because the system architecture ensures the amount of available energy at the beginning of each power cycle (see Section 3.4.2). Therefore, any variations in the input power will only impact the amount of time the device spends in charging C_{IN} and not in the energy available at the beginning of the power cycle. Second, the intermittently-powered systems considered in our work have the characteristic that $I_{EH} << I_{SYS}$ (see Fig. 3.4). Therefore, the effect of the power variation has negligible impact on the energy consumption characteristics of the device.

4.6 Discussion: Addressing inconsistency in intermittently-powered systems

As mentioned in Section 2.3.2, the semantic correctness of applications executing in intermittently-powered systems depend on the validity of the restored checkpoint after wake up. A checkpoint can become invalid or *inconsistent* with the current state of the system in two scenarios. The first scenario occurs if a task that cannot be distributed across two power cycles gets power-interrupted. These tasks can be perceived to be analogous to "critical" sections in traditional programs for which interrupts are disabled. Examples of such tasks include non-idempotent actions such as I/O operations, NVM accesses, and communication [57]. A power loss during communication can leave the IoT device in an inconsistent state unless proper re-initialization of the protocol is performed before recommencing data transmission. For example, in systems using WiFi, the transport layer security (TLS) parameters such as session ID, session ticket, encryption keys, *etc.*, that reside on the `stack` (and thereby a part of the checkpoint) cannot be reused in a subsequent power cycle as they need to be re-configured again to prevent malicious attacks on the system. Likewise

for BLE, the link layer connection parameters that set the channel map and the seed value for the channel hop algorithm cannot be reused from a checkpointed state. Similarly, certain digital peripherals need to be configured after wake-up before they can become useful. Unfortunately, critical tasks in intermittently-powered systems cannot disable (or stop) a power-interrupt from happening. Therefore, hardware-software techniques have to be employed to ensure the complete execution of a critical task before the inevitable arrival of the power-interrupt.

The second scenario is an artifact of the time spent by the system in the `OFF` state in between consecutive power cycles. In energy harvesting systems with little notion of time, continuing executions from the saved snapshot on power restoration is not always functionally correct. This is because, the checkpointed state might become stale, and hence become invalid on wake-up. For example, during the time in which the system recovers energy to wake-up again, the collected data samples may have already become stale and therefore should not be used for computations to follow. Further, the length of time taken for charging is unpredictable as it depends on the strength and availability of ambient source, both of which vary depending on multiple uncontrollable factors. A plausible solution to the problem is to execute time-sensitive tasks in an atomic manner. Atomic execution will ensure that sampled data is also used in a timely manner that would result in correct inferences.

The notions of critical tasks and atomic execution are built into our proposed algorithms, *eM-map* and *Energy-Align*. *Energy-Align* executes a function at runtime only if it can be completed within the power cycle. It utilizes a pre-characterized energy score for this purpose. Critical tasks within the program, when constructed as functions could be subjected to *eM-map* and *Energy-Align* and be completed before a power loss. For atomicity purposes, the pre-characterization can be performed to just find the energy consumption of executing the atomic task. Our proposed *eM-map* algorithm extends beyond this basic requirement and further reduces the energy consumption of the task by finding the optimal memory map. For *eM-map*, the atomic tasks could be encompassed into a function that is input into the algorithm. The re-

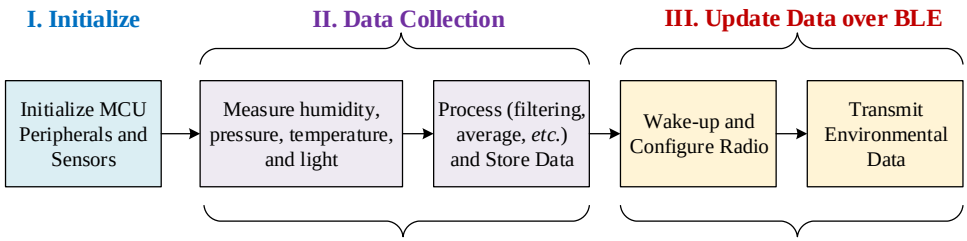


Fig. 4.11. Program flow used in case study for environmental monitoring

sponsibility of identifying atomic tasks and partitioning the program into constituent functions lies with the programmer. However, partitioning and creating functions is insufficient for atomic execution since *eM-map* does not guarantee a single power cycle execution. If the function execution energy exceeds the energy available at the beginning of a power cycle, the *eM-map* algorithm decides to split the function execution across power cycles. To avert such an outcome, the system has to provide sufficient input energy (adaptively or otherwise).

4.7 Case Study: An environmental monitoring edge device

In this section, we describe the case study of an IoT edge device, which executes a real application that monitors environmental conditions such as ambient temperature, humidity, pressure, light, *etc.*, performs computations on them, and also transmits the data over Bluetooth Low Energy (BLE). The program flow for the application is shown in Fig. 4.11 and consists of three segments (colored differently) that are to be performed *atomically*, *i.e.*, without any power interruptions disrupting their execution. Atomicity is an essential criteria for applications that are to be run on intermittently-powered systems for functional correctness, consistent execution, and efficient utilization of harvested energy [57]. Consider the three segments shown in Fig. 4.11, namely, the initialization of MCU peripherals and sensors, collection of environmental data, and transmission of the sensed data over BLE. Initialization involves configuring the MCU peripherals (such as ADC, SPI, I2C, *etc.*) that communicate

with the sensors along with configuring the sensors themselves (*e.g.*, frequency of sampling, setting output voltage, *etc.*). If the initialization step is not performed in an atomic manner and a power interruption occurs, it would result in energy being wasted as the step needs to be repeated in the subsequent power cycle. The second segment, collecting the environmental data, involves sampling the sensor, performing computations that involve filtering, statistical calculations, *etc.*, and converting the sensed data into comprehensible units of measurement. Many present-day sensors communicate with the MCU over synchronous buses such as SPI and I2C. The communication in such a system is via a command-response mechanism wherein commands issued by the MCU initiates operations such as sensing, transmission, *etc.*, in the sensor, and the sensor responds to the MCU with data or associated messages. Losing power and checkpointing the state in between such a transaction would force the MCU into an inconsistent state upon recall. This is because while the state within the MCU can be checkpointed, a checkpoint of the external sensor's state is not possible with the state-of-the-art. Hence, appropriate commands to the sensor have to be re-issued again in the subsequent power cycle to avoid the inconsistent state. Additionally, utilizing sampled data from multiple power cycles might not be acceptable as the interval between consecutive power cycles is dependent on the unreliable energy source, which may cause the data to become stale. Hence, the step of collecting data has to be performed atomically. Finally, the last step that has to be performed in an atomic manner is the transmission of data. In addition to the aforementioned inconsistencies that occur for communication parameters across power cycles, communicating data wirelessly is an energy-intensive operation and therefore, waking up the radio multiple times to transmit the same set of data across multiple power cycles is an overhead best avoided.

For the case study conducted in this work, we perform *Energy-Align* at the segment boundaries to enforce atomic execution of each segment as shown in Fig. 4.12. Initialization and data collection steps happen in the same power cycle atomically before the system is forcefully shutdown to gather enough energy for BLE transmis-

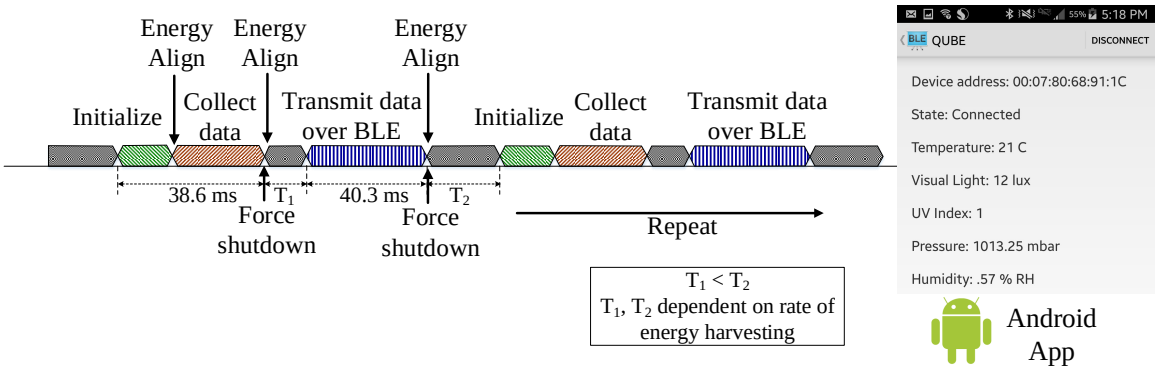


Fig. 4.12. Function execution time-line for the environmental monitoring application

sion. Following this, the collected data is transmitted also in an atomic manner. The procedure is repeated as and when energy is available. To visualize the transmitted data and verify the functionality of the IoT edge device, we designed an Android mobile application and ran it on a Samsung Galaxy S5 mobile phone whose GUI is also shown in Fig. 4.12.

4.8 Related work

In this section, we first discuss the related work corresponding to inconsistent checkpoint states in periodic checkpointing schemes, software techniques for executing applications in an atomic manner, and then give an overview of non-volatile processors targeted for intermittently-powered systems.

4.8.1 Inconsistency due to periodic checkpointing

As discussed in Section 4.6[1], the location of the program at power loss has direct consequences to the correctness of the application and affects the validity of the check-

[1]The reader is suggested to refer Sections 2.3.2 and 4.6 prior to the study of this section

point in the subsequent wake-up cycle. Research conducted by Lucia et al. [57–60] consider scenarios that could result in an inconsistency between the checkpointed state of the system and the state it wakes up to. They consider a software design that has predefined checkpoint locations in the program (akin to Mementos [34] discussed in Section 3.8). When the program execution reaches such a location, a checkpointing operation is performed, and the program continues to execute with the remaining energy (in somewhat a greedy manner) until power is lost or until the next checkpoint location is reached. If power is lost in between two checkpoint locations, the system rolls-back to the last saved checkpoint in the subsequent wake-up cycle. However, if the program alters any data (state) of the non-volatile memory during the (greedy) execution phase after it has performed the checkpoint, then the restore operation would roll-back to a state that is inconsistent with the checkpointed state causing the application to proceed in an unintended direction. This scenario is an artifact of the fact that it allows (greedy) execution of the program even after a checkpoint operation without buffering enough energy to guarantee another checkpoint operation. QuickRecall, on the other hand waits for enough energy to be buffered before proceeding execution and thereby, avoids greedy execution (as shown in Fig. 3.2).

4.8.2 Software techniques for atomic execution

Software techniques that propose to execute critical tasks of a program in an `atomic` manner such that it is not interrupted by a power loss have been explored. Such techniques ensure the semantic correctness of applications executing in intermittently-powered systems. DINO [57] is such a compiler that partitions the program into constituent tasks utilizing programmer annotations. DINO analyzes the task boundaries that are annotated by the programmer and performs an analysis of the costs of checkpointing and restoring operations to emit warnings and suggestions for minimizing the overhead.

For programs already partitioned into tasks, Dewdrop [61] adjusts the input energy to allow completion. Dewdrop is an energy-aware run-time that lets the system replenish its available energy until it is sufficient to execute the particular task. It learns about the amount of energy required per constituent task in an iterative manner by increasing and decreasing the amount of energy so as to converge upon the desired amount. Another approach is to utilize an energy-buffer in the system that would provide the necessary additional energy, if required, to complete the task even if a power interrupt occurs. Ref. [50] performs such an approach and an energy analysis is performed off-line to calculate the amount of buffer energy.

Other techniques such as Ref. [62] assume a greedy checkpointing and execution scheme and propose techniques for making execution consistent.

4.8.3 Non-volatile processors

An orthogonal approach for retaining state in IoT systems with unreliable power supply is to perform the checkpoint operation entirely in hardware using non-volatile processors (NVPs). These processors are designed using memory elements (flip-flops and RAM) that are augmented with a non-volatile storage. The memory elements automatically checkpoint the volatile state on power loss and restore it on the subsequent power cycle without any software support. Kothari et al. [63] simulated an Intel P4 processor augmented with nanomagnetic devices to enable checkpointing in large-scale systems. Yu et al. [64] simulated an 8-bit microcontroller whose volatile memory elements were integrated with floating gate elements for checkpointing. Wang. et al. [4] fabricated a nonvolatile processor that used ferroelectric flip-flops to make the processor core non-volatile. Bartling et al. [5] and Khanna et al. [6] created an 8 MHz processor having all flip-flops to be non-volatile using ferroelectric capacitors. Sakimura et al. [11] fabricated a completely non-volatile microcontroller based on the MSP430 architecture, integrated with nonvolatile flip-flops (using MTJs) and a 64 kB MTJ based RAM. Singhal et al. [8] fabricated a single cycle

16 MHz microcontroller with ferroelectric elements. Onizawa et al. [65] simulated an ARM-based non-volatile processor that uses STT-MRAM as the non-volatile element. Liu et al. [13] fabricated a 100 MHz non-volatile processor that utilizes non-volatility provided by ReRAM. They employ a non-volatile SRAM consisting of a resistive memory element, that acts as an SRAM under normal operation.

Researches have also been conducted for exploring the architecture of non-volatile processors. Ma et al. [66, 67] explores different policies and provides a comparative-analysis on the different kinds of non-volatile processor architectures (using ferroelectric elements). Non-volatile processors reduce the overhead in storing and recalling the state as no instructions are required to be executed for data transfer. Solutions in Refs. [5, 11, 13] can bypass boot procedures and execute the next instruction on wake-up. For other NVPs, the SRAM contents still need to be checkpointed and software techniques to reduce the checkpoint size have been explored [68–71]. However, the initialization step that includes configuring the external sensors have to be performed again via a software boot-up method before resuming application code execution. Similarly, a software strategy is required for NVPs to enable atomicity in application execution. Like traditional processors, NVPs also suffer from aforementioned problems of stale data, false execution states, inconsistent communication states, and communication overhead arising due to program segments being broken and spread over multiple power cycles. Recent research in NVPs have tried to address the initialization overhead of peripheral registers by making them non-volatile [72]. However, the inconsistency in communication and the need for re-performing the protocol's necessary hand-shaking still exists in these NVPs. Chien et al. [73] proposed an ReRAM-based non-volatile processor to partially address the inconsistency issue. Their NVP utilizes a programmable restore point, to which the processor can wake-up to. In case the power interruption happens in between an IO communication (such as UART, SPI, *etc.*), utilizing the programmable restore point would roll-back to the location corresponding to the beginning of data transmission. Such an architecture improves the checkpointing support from hardware while still being able to provide

the programmer with the control of partitioning the program into tasks. However, the NVP allows greedy execution and therefore is susceptible to inconsistent checkpoints and energy wastage due to partial transmission. Therefore, even though NVPs help reduce the checkpoint and restore overheads, systems utilizing them would require a well-designed software architecture to enable atomicity and resolve the consistency issues in application execution.

At the time of composing this article, none of the above fabricated NVPs are commercially available off the shelf. Therefore, we utilize the MSP430FR5739 MCU that is embedded with FeRAM (instead of Flash) and based on Texas Instruments' MSP430 architecture similar to [7, 9].

4.9 Summary of contributions

In this chapter, we proposed techniques for performing energy aware memory mapping of program sections in hybrid FeRAM-SRAM MCUs used in intermittently-powered IoT edge devices to retain the reliability benefits of non-volatile FeRAM while performing as efficiently as SRAM. To this end, we defined a one-time characterization technique, *eM-map* that finds the optimal memory maps for constituent functions of a program across the hybrid FeRAM-SRAM memory. We also proposed a technique, *Energy-Align*, that performs proactive shutdown to align function and power cycle boundaries, thereby achieving energy and performance benefits. Our implementation using the MSP430FR5739 MCU demonstrates a speed-up of up to `2x` (`1.6x` on average) and energy reduction of up to `45%` (`30%` on average) compared to a state-of-the-art baseline solution.

5. SLEEP MODE VOLTAGE SCALING: ENABLING SRAM DATA RETENTION AT ULTRA-LOW POWER IN EMBEDDED MCUs

In this chapter, we consider IoT devices that are battery-powered but operate intermittently transitioning between active and idle modes. The work-load profile of such devices are characterized by long continuous durations of sleep interjected by short bursts of activity, during which the MCU performs the intended task. Due to the intermittent nature of operation, the sleep mode energy consumption dominates the overall power consumption. This chapter proposes a new ultra-low power sleep mode for embedded MCUs, which reduces sleep-mode power consumption by performing extreme voltage scaling of its supply voltage.

5.1 Chapter overview and contributions

As mentioned in Chapter 2, the dominant component in a system's total energy consumption is its idle-mode energy consumption. Even though MCUs provide multiple *shallow* and *deep* sleep modes to counter the sleep-mode power consumption, they still suffer from one or the other drawbacks related to energy-efficiency, wake-up latency, and lack of data retention. Shallow sleep is sub-optimal from a sleep-mode power consumption perspective as the MCU stays powered on to retain the state information (consisting of the MCU registers and the contents of on-chip SRAM) during sleep. Whereas, deep sleep involves a significant energy and time overhead for copying the volatile state information to and from the non-volatile memory before entering sleep and after waking up from it. To address this issue, this chapter proposes a new ultra-low power sleep mode for MCUs that is as good as deep sleep in terms of sleep-mode power consumption, but still preserves the contents of SRAM,

thus avoiding the energy-expensive data transfer overhead. The key insight behind the proposed sleep mode is the observation that the minimum voltage required for SRAM data retention is often much lower (by as much as `10x`) than the minimum operating voltage of the MCU. By lowering the supply voltage when the MCU is in sleep mode to just above the SRAM data retention voltage (we call this extreme voltage scaling[1]), we demonstrate that dramatic reductions in sleep mode power consumption can be obtained. Additionally, we propose and demonstrate the use of energy harvesting from commonly found on-board sensors in IoT devices (such as a photodiode used for sensing light) for supplying the required scaled voltage for the MCU.

Specifically, this chapter proposes and demonstrates a novel, lightweight on-chip SRAM data retention scheme, called Hypnos, for embedded MCUs. Hypnos is a HW/SW approach that significantly reduces the power required for *in-situ* data retention through the use of extreme supply voltage scaling in sleep mode. We use Hypnos to design, implement, and evaluate a new ultra-low power sleep mode, LPM_H, for the TI MSP430G2452 MCU. In our experiments, the MCU draws only 26 nA when in LPM_H, which is `4x` lower than any existing sleep mode of this MCU that preserves SRAM data. Further, we propose the use of a light sensing photodiode as an energy harvesting source to supply power to the MCU during LPM_H, which eliminates (almost entirely) the power overheads associated with performing voltage scaling. We demonstrate that utilizing the photodiode for power supply reduces the LPM_H current consumption to only 1 nA, which is over `100x` lower as compared to the conventional low power mode of the MSP430G2452 MCU. Finally, we show that, for a typical wireless sensing application, the use of the Hypnos scheme translates to a substantial reduction in the average power consumption of the *entire system* (by `6.45x` in our configuration), compared to a well-optimized baseline. Operating at such extremely low power raises the possibility of perpetual system operation by harvesting energy from ambient sources.

[1]Conventionally, voltage scaling is done when the MCU is in active mode and is accompanied by a reduction in the MCU clock frequency. In contrast, here we are talking about scaling the MCU's supply voltage when it is in sleep mode.

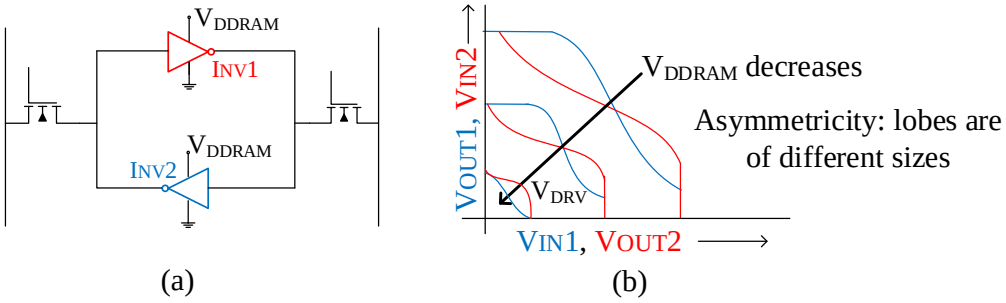


Fig. 5.1. (a) An SRAM cell (b) Voltage dependent latching in an SRAM cell

5.2 Preliminary study: SRAM data retention at scaled MCU supply voltages

This section initially provides the underlying reasons for SRAM data retention at low voltages and then describes an experiment that demonstrates the key insight that motivates our work.

5.2.1 SRAM data retention at low voltages

An SRAM cell is made up of two back-to-back inverters as shown in Fig. 5.1(a). The back-to-back connection creates a positive feedback loop that reinforces the value written into the cell. This characteristic has a direct dependence on the V_{DDRAM} of the two inverters. Fig. 5.1(b) shows the variation in inverter characteristics of the SRAM cell as a function of V_{DDRAM}. As V_{DDRAM} decreases, the reinforcing action loses its strength until it collapses and cannot regenerate the stored value. The lowest voltage at which the cell can retain the stored value is called the data retention voltage (V_{DRV}) or the hold voltage and is usually lower than its operating voltage. Characterizing the V_{DRV} of SRAM cells has been a well-studied topic [74–76]. The ability of SRAM cells to retain data at a lowered supply voltage has also been exploited by circuit designers to reduce the leakage power of SRAMs when idle [77–80]. In order to verify this observation at a system-level, we conducted an experiment using a commercially available MCU and decreased its supply voltage to find its V_{DRV} (in Section 5.2.2).

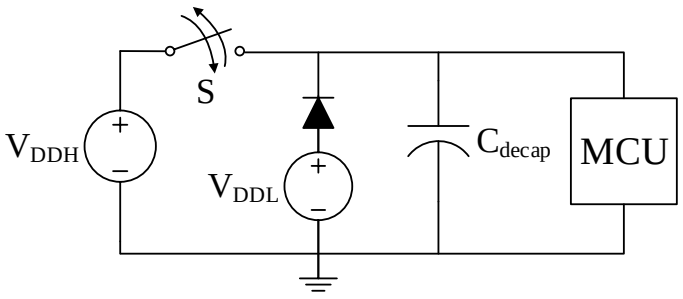


Fig. 5.2. Data retention experimental setup

Strong and weak states in SRAM cells

Due to the asymmetric nature of the butterfly curve (as illustrated in Fig. 5.1 (b)), SRAM cells have an affinity to one of the two logic levels when powered up. The area of the lobes also denote the stability of the SRAM cell for each logic value. An SRAM cell that has a larger lobe for logic level 0 is referred as a strong-0 cell and likewise for a strong-1 cell. Therefore, a larger difference in potential is required to flip the cell from its stronger logic value as compared to flipping the cell from its weaker logic value. As the supply voltage is lowered, the asymmetric nature of the butterfly curve is more or less preserved. Therefore, when the MCU is powered up, the default value will correspond to its stronger logic level [81]. In this work, we also verified this observation in the context of powering up a microcontroller with on-chip SRAM.

5.2.2 Motivational study

The experimental setup is shown in Fig. 5.2. V_{DDH} is kept at a constant 3.3 V, while V_{DDL} is varied for the experiment. The procedure of the experiment is described as follows. The switch S is initially in the closed state, and the microcontroller receives a supply voltage of V_{DDH}. 128 bytes of data is written into a predefined location in the on-chip SRAM of the MCU. The data values written are chosen corresponding to the weaker logic value for each bit, *i.e.*, if a cell is strong-0, a logic value of 1 is written to it and vice versa. The strong and weak value for each cell is known *a priori*

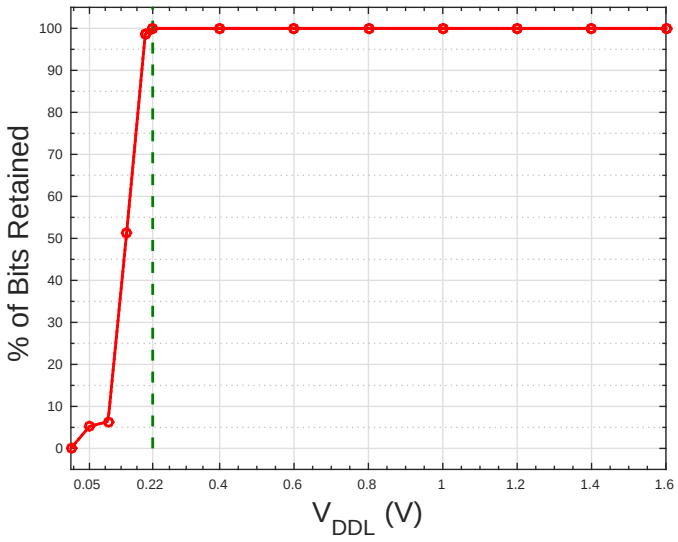


Fig. 5.3. SRAM data retention with varying V_{DDL}

by reading its default value at power up. Then, the switch S is opened for exactly 10 seconds (t_{open}) using a timer and closed again. The data is then read back and compared to the written data. Writing the weaker value ensures that even worst case data flips are detected. This procedure is repeated for different values of V_{DDL}. The MCU used for this experiment is the TI MSP430G2452 [82], which does not have an internal power management unit[2] (PMU). The results are shown in Fig. 5.3.

As Fig. 5.3 shows, the on-chip SRAM retains 100% of the written data for V_{DDL} up to 220 mV, henceforth referred to as V_{DRV}. Observe that when the supply voltage is less than V_{DRV}, the data retention degrades rapidly. This is due to the fact that an increasing number of SRAM cells flip to their stronger value, and thus become unreliable for voltages lesser than V_{DRV}. Note that the normalized BER degrades to a value close to 0% because each SRAM cell always resets to its stronger logic value upon power up.

We repeated the above experiment for 20 different MSP430G2452 MCUs and plot their V_{DRV} in Fig. 5.4. As Fig. 5.4 shows, the V_{DRV} of the 20 MSP430G2452

[2]The impact of an internal PMU on the experiment is discussed in Section 5.7.2

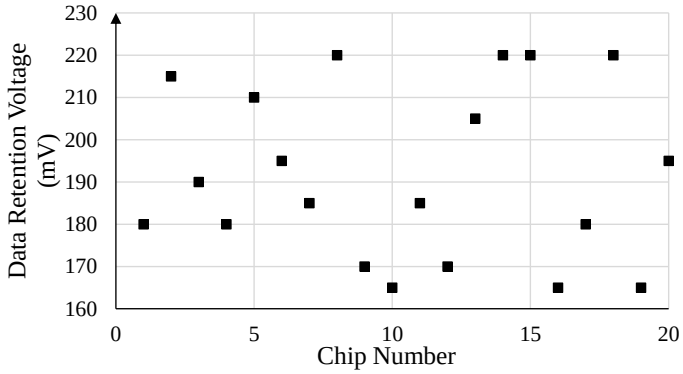


Fig. 5.4. V_{DRV} for 20 different MSP430G2452 MCUs

MCUs range from 165 mV to 220 mV that can be attributed to variations in the manufacturing process of the MCUs. Hence, we choose the highest voltage for which 100% data retention is observed across the 20 different MCUs as the chip DRV for the MSP430G2452 MCU. An alternative approach could be to perform an *in-situ* characterization for the specific MCU chip that is used in a system. If such per-chip characterization is not feasible, a simpler alternative would be to simply guard-band the DRV to counter variation-induced deviations, as we discuss in Section 5.7.1.

The experiment is repeated once more to verify data retention by varying t_{open} from 10 s to 240 s (and, in a subsequent experiment, setting t_{open} to 24 hours) by setting the V_{DDL} to a constant 220 mV. Results confirm that SRAM data is retained without even a single bit flip for all t_{open}. We conclude that at $V_{DRV} = 220$ mV, the on-chip SRAM cells of the MSP430G2452 MCU can retain data for practically infinite time. We define this voltage to be the chip DRV of the MCU. This observation that microcontrollers can retain SRAM data at much lower supply voltages than the specified operating voltage motivates us to investigate and propose a new LPM for MCUs.

Table 5.1.
Dependency of SRAM retention on ambient temperature at various voltages

Temperature (°C)	220 mV	260 mV	300 mV	310 mV	320 mV
23	Yes	Yes	Yes	Yes	Yes
40	Yes	Yes	Yes	Yes	Yes
60	No	Yes	Yes	Yes	Yes
70	No	No	No	No	Yes

Yes denotes 100% SRAM retention, a *No* is indicative of even a single bit-flip

5.2.3 Impact of temperature on chip DRV

In Section 5.2.2, we defined chip DRV to be the smallest supply voltage at which 100% data retention is observed across the entire SRAM memory in the chip, and for the MSP430G2452 MCU, V_{DRV} was observed to be 220 mV. The theoretically estimated DRV of an SRAM cell is 50 mV [74] for a 90 nm technology. However, process variations cause the actual DRV of an SRAM cell to diverge from the estimated value [83], thus resulting in a much higher value for the chip DRV. To gauge the impact of temperature on the V_{DRV} of MSP430G2452, we conducted an experiment wherein SRAM retention was verified after varying the ambient temperature of the system in LPM_H. The experimental methodology is similar to Section 5.2.2 but with the addition of a control knob for tuning the ambient temperature. Table 5.1 tabulates the results showing the impact on SRAM data retention for different supply voltages (V_{DDL}) as the ambient temperature of the system is varied. Even a single bit flip is considered to be an error and is marked as `No` implying that SRAM retention is not guaranteed for that operating condition.

As can be seen, the data residing in SRAM is not reliably retained for a chip DRV of 220 mV when the ambient temperature is greater than 40 °C. The table also shows

that for an increase of 100 mV in the chip DRV, 100% of SRAM data retention can be achieved for ambient temperatures up to 70 °C. Therefore, the chosen value of V_{DDL} has to be the chip DRV, which includes a guard-band that can reliably retain the SRAM data according to the environmental conditions (such as temperature) at the deployment location. However, note that over compensating for temperature variation with a higher voltage for V_{DDL} increases the current consumption of the MCU in LPM_H and hence, will affect the amount of reduction in energy consumption that is achieved.

5.2.4 Discussion: Impact of technology scaling on chip DRV

The MSP430G2452 used in this work is fabricated using the 130 nm technology node [84]. It is well known that as technology scales down, the amount of leakage current also increases. Consequently, the chip DRV also increases with scaling as more energy is required to prevent data bits from flipping. The authors in [85] show that chip DRV increases by almost 100 mV with each successive technology node. Further, as technology scales, process variations have a bigger impact leading to an increase in the DRV [75]. Circuit techniques could be employed to improve the DRV [76], but that does not mitigate the issue completely. Hence for scaled technology nodes, the V_{DRV} has to be set at a much larger voltage.

5.3 HYPNOS architecture and design

In this section, we describe the hardware and software architectures for HYPNOS in detail.

5.3.1 Hardware architecture

Conventional implementations of LPM in microcontrollers primarily halt the clock subsystem and the microcontroller core, thereby stalling computations and entering

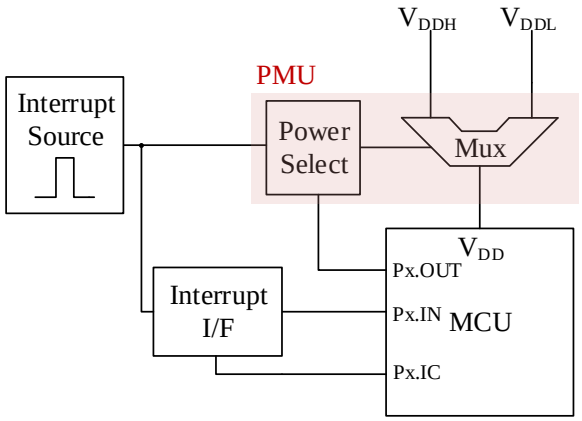


Fig. 5.5. HYPNOS hardware architecture

the low power mode. The system then waits for an interrupt upon which the LPM is exited. Often, an internal or external real time clock (RTC) is used to wake the system up with a periodic interrupt. On receiving the interrupt, the corresponding interrupt service routine (ISR) is executed, following which the program may remain in the active state or re-enter the sleep mode. Any new LPM should be identical to the conventional LPMs with respect to the aforementioned features exhibited and only differ in their latency and power overheads. We propose and define a new sleep mode (henceforth referred to as LPM_H) that decreases the static power consumption of embedded microcontrollers in idle mode by supplying a lower voltage while still retaining data. Fig. 5.5 shows the HYPNOS hardware architecture.

As discussed in Section 5.2, data can be retained even when supply voltage is as low as 220 mV for the MSP430G2452 MCU. Therefore, the primary design decision for HYPNOS is to enable the scaled supply voltage (V_{DDL}) to power the system in idle mode. The V_{DDL} could be derived from V_{DDH} itself, or it could be a stand alone supply. A discussion on the same can be found in Sections 5.5 and 5.7. An external PMU, which consists of a power selection unit (PSU) and a multiplexer is designed to address the dual voltage requirements during active and idle state. The design of the power selection unit is critical to the PMU architecture. Identical to conventional LPMs, the MCU has to wake up from LPM_H as soon as an interrupt is

received. Likewise, the PMU needs to supply a voltage of V_{DDH} continuously to the MCU in the active mode and make a successful transition to V_{DDL} upon initiating LPM_H. The program executing on the MCU controls the different power-modes and alternate between them as required by the application. Since the PMU lies external to the MCU, a general purpose input/output (GPIO) pin is required to administer control to it. `PxOUT` (shown in Fig. 5.5) is utilized to convey the state of the MCU to the PMU. When the MCU is active, `PxOUT` is pulled high and the V_{DDH} selection is reinforced. To enter LPM_H, `PxOUT` is pulled low by the MCU, which causes the multiplexer output to switch to V_{DDL}. The PSU keeps the MCU in LPM_H until an interrupt is received or an external power-up event occurs. An interrupt prompts the PSU to swap the multiplexer output back to V_{DDH}, thus putting the MCU into active mode.

External interrupts are generally characterized by pulses of extremely short duration. The interrupts are not only used to wake the MCU up from a sleep mode, but also to trigger an ISR that executes some application task. In conventional LPMs, the interrupt triggers a transition to active mode and gets registered at the appropriate GPIO by the software. The time taken for this transition is minimal, and is usually in the order of a few microseconds to milliseconds depending on the LPM in use. In addition, the ports of the microcontroller receive a supply voltage, which is equal to V_{DDH} and therefore, the application software can register the interrupt from the associated GPIO instantaneously. In comparison, LPM_H reduces the microcontroller supply voltage to V_{DDL}, and therefore the port logic is defunct and unreliable, which may result in an interrupt being missed. Therefore, a new interrupt interface is envisaged whose primary function is to latch the interrupt that is received when the MCU is in LPM_H (shown in Fig. 5.5). The modifications required for registering the interrupt by the application software are discussed in the following subsection. Note that all the interrupts of the microcontroller need not have an interrupt interface. Only those interrupts that are designated to wake up the microcontroller from

LPM and cause a transition into the active mode need to be implemented with the interface. We classify such interrupts as wake up interrupts (`WInts`).

Fig. 5.5 shows that an additional microcontroller GPIO (`PxIC`) needs to be utilized so as to control the interrupt interface. `PxIC` is used to reset the interface after the latched interrupt has been registered by the software. Depending on the number of interrupts that have wakeup capability, the designer could re-use `PxIC`. Mathematically, for n `WInts`, a minimum of $log_2 n + 1$ GPIOs are required for controlling the interrupt interface. Thus, a total of $log_2 n + 2$ GPIOs are needed to implement HYPNOS.

5.3.2 Software architecture

The software architecture for HYPNOS is slightly different from conventional LPMs. The software flow for entering a traditional LPM involves configuring the registers of the GPIOs and enabling the appropriate `WInts` before proceeding to shut off the clock module and other sub-systems. For example, to enter LPM4 in the MSP430G2452 MCU, first the GPIO directions and signals are set such that the pin leakage is minimized. Secondly, the `WInts` are enabled and lastly, the clock subsystem is switched off, halting the computations and forcing the system into LPM4. Subsequently, when an interrupt is received, the microcontroller, the clock subsystem, and the system peripherals have to be re-initialized in that order, after which the application execution can commence.

By design, HYPNOS retains only the data present in the SRAM. The data pertaining to the processor registers (*i.e.* the PC, SP, SR, *etc.*) and GPIO configurations are not retained *in-situ*. Hence for applications that require to resume computations upon wake-up, the processor registers, the state of the microcontroller, and that of its peripherals need to be preserved. In HYPNOS, we propose to achieve this by checkpointing the processor registers onto SRAM before entering LPM_H.

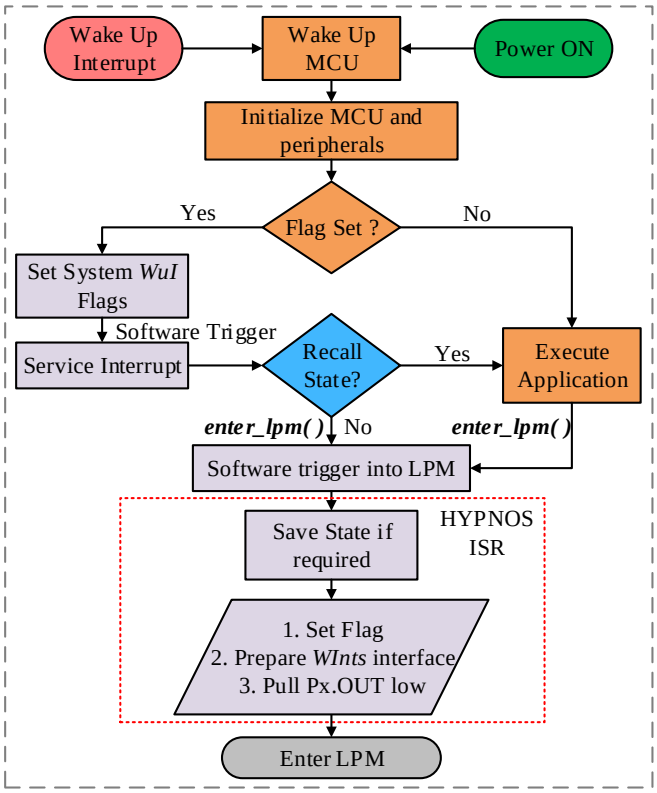


Fig. 5.6. HYPNOS software architecture

Fig. 5.6 illustrates a general software flow for implementing HYPNOS. It tries to address two types of applications. The first set consists of periodic sensing applications wherein the system wakes up on an interrupt and executes an ISR that does the required sampling and computation. Once the execution completes, the system goes back into LPM_H and sleeps until the next interrupt arrives. Examples of periodic sensing applications are *in-vivo* glucose monitoring [35], ambulatory urodynamics [37], environmental sensing, *etc.* The second kind of applications are event-driven and conserve power by waiting for the event to occur in a low power mode. Once the event is detected, the system wakes up and executes a sequence of operations specific to the application before re-entering the low power mode. Examples of such applications are structural health monitoring systems, disaster management systems like flood detection, forest fire warning systems, *etc.* [36].

An MCU can wake up either due to a normal power up event or due to the arrival of a `WInt`. Fig. 5.6 shows the HYPNOS software flow for both these wake-up (boot-up) scenarios. Once the MCU is powered up, the software routine re-initializes the microcontroller and its peripherals. Then, it checks a flag (`hypnosflag`) to verify if it is waking up from LPM_H. Note that, the `hypnosflag` is stored in SRAM and its location in the memory is chosen such that it has a default (strong) value of 0x0. If the flag is not set, the application continues with the program execution just like a normal power-up event. Otherwise, the HYPNOS software routine proceeds to evaluate the values of all the `WInts` in the system to identify the particular one that triggered the wake-up. This is explained as follows using an example. Consider a system having two `WInts`, `IntA` and `IntB`. If `IntB` is triggered, then the interrupt interface latches the value and wakes up the MCU. Once the MCU wakes-up, the HYPNOS software flow sequentially compares the values of `IntA` and `IntB` according to a predefined order of priority. Once `IntB` is identified to have caused the wakeup, a software trigger is issued that executes the appropriate ISR for `IntB`. The implementation of the software trigger can vary from MCU to MCU and, for the MSP430G2452, a simple write to the internal GPIO interrupt register triggers the ISR execution. Thus in HYPNOS, the `WInt` that triggered the wakeup event gets registered and the corresponding ISR is executed. Depending on the application, the program can either return to the saved state and resume computations or re-enter LPM_H.

The data in processor registers, GPIO configuration registers, and GPIO values are not retained when the microcontroller is supplied with a voltage V_{DDL}. Therefore, a checkpoint operation has to be performed before entering LPM_H. As Fig. 5.6 shows, HYPNOS defines a function `enter_lpm()` that triggers a software interrupt. The program context gets pushed onto the stack upon entering the HYPNOS ISR. Then, the processor registers (whose number depends on the application) get checkpointed onto the SRAM for retention. The GPIO configurations are restored while waking (booting) up in the `Initialize MCU and peripherals` step. The specific cases where GPIOs change their values or configurations in the course of program execution are

handled separately. A turn-key solution would copy all the register values into the SRAM, thus making the solution application-agnostic. However, such an approach will incur additional latency and reduce the total amount of SRAM memory available for program execution. Another approach is to utilize the programmer's knowledge about the application to determine a subset of registers that may change state during the course of program execution, and checkpoint this subset only while initializing the rest of the registers in the `Initialize MCU and peripherals` step. This approach is application-specific and will incur lesser memory and latency overheads. Therefore, a trade-off exists in saving the state of the GPIO registers in regard to the latency overhead and amount of SRAM memory that is required, and programmer involvement. After checkpointing, the ISR finally proceeds to set the `hypnosflag` and the `WInt` interface control signals, and toggles the `PxOUT` to select V_{DDL} as the power supply. Thus, the microcontroller enters LPM_H.

5.4 Sleep mode voltage scaling

As discussed in Section 5.3, Hypnos requires two different voltages to supply power to the system. Conventionally, microcontrollers are powered with only a single power supply. Including another power supply introduces additional complexity in system design and implementation. Hence, our implementation consists of a single voltage source corresponding to V_{DDH} while V_{DDL} is either derived from it or generated using an energy harvesting source. Both of these are described in detail below.

5.4.1 V_{DDL} generation from V_{DDH}

Generation of V_{DDL} from V_{DDH} is performed by a voltage converter block, whose architecture is chosen such that the power overhead of implementing LPM_H is minimized. For example, the required V_{DDL} could be generated using a reference constant voltage source (V_{REF}) or by using a voltage divider. At the time of composing this

article, an off-the-shelf V_{REF} module that supplies V_{DRV} draws 250 nA as quiescent current. To avoid this, we implement the voltage converter block using a voltage divider that consists of a resistor on the high-side in series with the microcontroller, which provides the voltage and current that is required by the system to implement LPM_H. Note that the block is architected in such a manner that it does not consume power when the MCU is active. However in LPM_H, it consumes power equivalent to that of the MCU, which is considerably lesser due to scaling of the supply voltage.

5.4.2 V_{DDL} generation by energy harvesting

The V_{DDL} generation for HYPNOS is of considerable importance as it affects the idle mode power consumption and the overall energy consumption of the system. Using the resistive divider still impacts the idle mode power consumption due to the I^2R loss involved in V_{DDL} generation. An alternate method to generate the required V_{DDL} is to utilize some kind of energy harvesting technique from an ubiquitous source. A few options are described below. For applications that use RF for communication, harvesting energy from a dedicated RF source [86–88] or ambient RF signals [89] could provide sufficient power to meet the requirements of LPM_H. Another kind of energy harvesting source that could power the system in LPM_H are nanogenerators [90–92]. Nanogenerators convert mechanical energy to electrical energy and provide power in the range of microwatts to milliwatts. Light is another ubiquitous energy source that could be utilized for V_{DDL} generation. The choice of the energy harvesting source is governed by factors such as the deployment location and form-factor of the embedded system [2]. The location determines ambient energy availability while the form-factor determines the amount and rate at which the energy can be harvested (*e.g.*, it determines the size of a photovoltaic cell). However, note that since we propose to use energy harvesting only as a means to support LPM_H (and not for regular system operation), these constraints can easily be met.

(a) A well-lit office gets around 600 lux

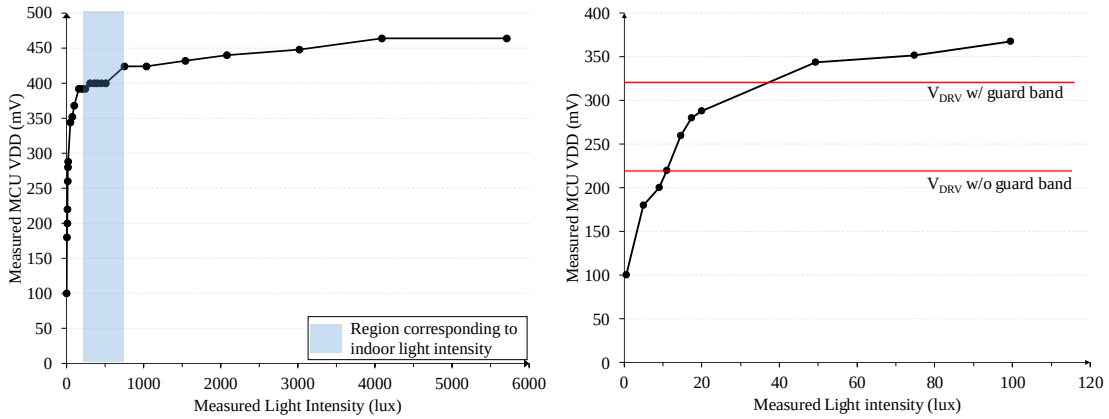
(b) SRAM data retained in near darkness (light intensity of 20 lux)

Fig. 5.7. Light intensity measurements in an office environment

Utilizing a light-sensing photodiode as an energy harvesting source for $\mathbf{LPM_H}$

Photodiodes are passive transducers that convert the incident light energy into electric current. They have a very small form-factor and are conventionally used to measure light intensity in various embedded applications. Previous work has shown that such a light-sensing photodiode could be utilized as an energy source to supply power to the real time clock (RTC) of an embedded system [25]. In this work we build on the same concept to eliminate the power overhead for V_{DDL} generation.

The architecture of HYPNOS is kept the same as shown in Fig. 5.5 except that the resistive voltage converter circuit is now replaced with a photodiode that is connected to the V_{DDL} port of the power selector unit. Therefore, once the MCU enters LPM_H, power to the MCU is supplied by the photodiode output. The photodiode acts as a current source and the voltage (V_{DDL}) that appears across the MCU is a result of the amount of load offered to the photodiode by a combination of the MCU and the voltage selector multiplexer. The amount of current generated by the photodiode depends on the intensity of light that falls on it. Light intensity is measured in lux and



(a) MCU VDD vs. light intensity

(b) Magnified x-axis to show minimum light intensity required for supporting LPM_H

Fig. 5.8. Variation of MCU VDD according to incident light intensity

a well-lit indoor office receives light intensity in the range of $300 - 700$ lux as shown in Fig. 5.7(a). As the light intensity varies, the output current from the photodiode also varies, which in turn affects the voltage supplied to the MCU. Fig. 5.8(a) shows this variation of V_{DDL} as a function of the incident light intensity. Fig. 5.8(b) is a magnified version of the same. As can be seen, the voltage across the MCU increases sharply at first for light intensities less than 200 lux, and then plateaus off for further increase in light intensity. The initial (almost linear) surge in the voltage can be attributed to the fact that the current generated by the photodiode has a linear dependence on light intensity. The plateau that follows is due to the fact that the open-circuit voltage (V_{OC}) of the photodiode is 470 mV. Therefore, as light intensity increases, the output voltage of the photodiode tends to and saturates at 470 mV.

Note that the light intensity at which the MCU receives the data retention voltage ($V_{DRV} = 220$ mV) was observed to be 11 lux. Also, observe that for light intensities greater than 36 lux, the photodiode is able to provide a stable supply voltage that includes the 100 mV guard-band required for accommodating variations in temperature. Further, we successfully verified 100% SRAM data retention in LPM_H (for over 10 hours) for a light intensity as low as 20 lux, which is near-darkness as shown

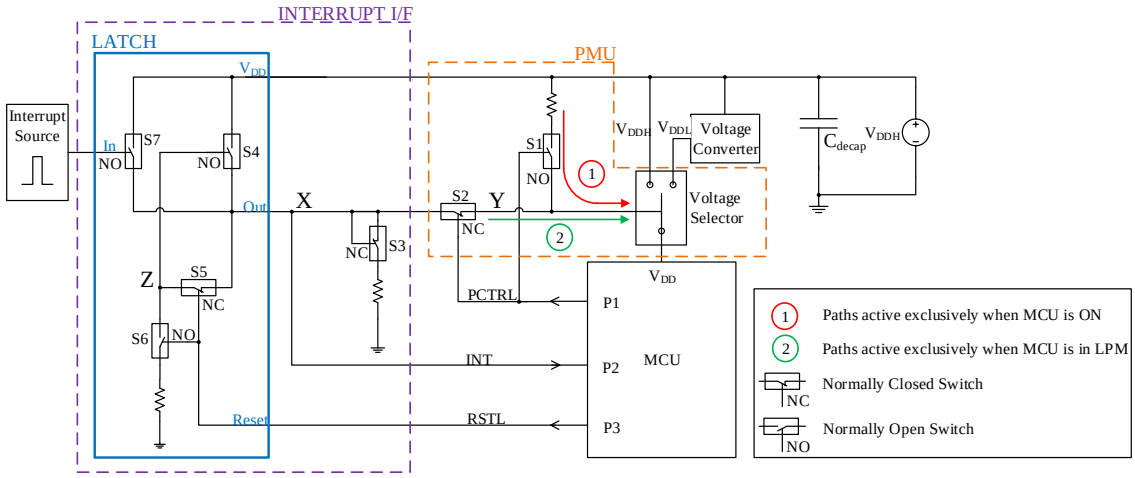


Fig. 5.9. HYPNOS hardware implementation

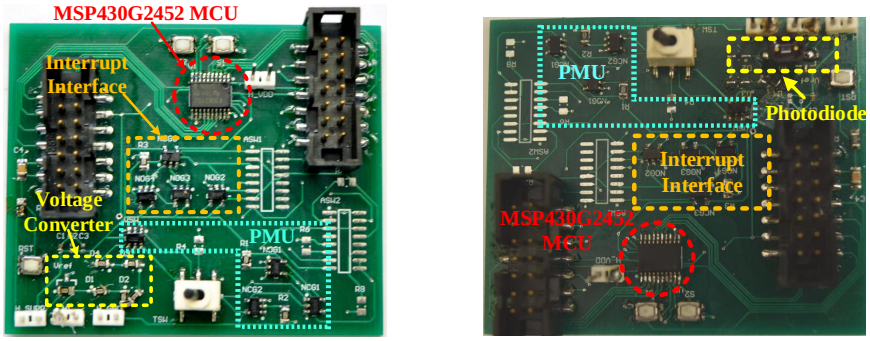
in Fig. 5.7(b). Finally, note that in Fig. 5.8(a), for the region corresponding to indoor light intensities, the voltage across the MCU is more than the guard-banded 320 mV. Therefore, embedded applications located in well-lit indoor offices can retain 100% of SRAM data.

5.5 Low power implementation

In this section, we explain our hardware implementation for HYPNOS and discuss the design decisions in detail. Fig. 5.9 shows the HYPNOS hardware circuitry in detail and Figs. 5.10(a) and 5.10(b) shows the experimenter boards that we designed and implemented. In particular, Fig. 5.10(b) shows our modified experimenter board with the Si1133 photodiode (from Hamamatsu) being employed as the energy harvester for V_{DDL} generation.

5.5.1 Power supply

V_{DDH}, which is the primary supply voltage for the system, is supplied from a battery. On the other hand, V_{DDL} is either derived from V_{DDH} using a voltage converter



(a) V_{DDL} generated from V_{DDH} using a voltage converter

(b) V_{DDL} supplied from a light sensing photodiode

Fig. 5.10. Custom HYPNOS experimenter boards

block or generated from an energy harvesting source as mentioned in the previous section. The other component within the power supply architecture shown in Fig. 5.9 is the decoupling capacitor (C_{decap}). Note that C_{decap} is placed before the voltage selector on the V_{DDH} rail and not on the microcontroller V_{DD} rail. The placement of C_{decap} is of considerable importance due to the charging overhead it may present while transitioning from LPM_H to active mode. This overhead has a direct impact on the wakeup latency of HYPNOS. Hence, to minimize the overhead, the placement of C_{decap} is performed in such a way that avoids redundant charge-discharge cycles. In addition, the current consumption in LPM_H is stable since the MCU is idle and does not perform any operation. Thus, a C_{decap} on the V_{DDH} rail would suffice.

5.5.2 Power management unit

The primary function of the external PMU in HYPNOS is to effectively switch the MCU's power supply between V_{DDH} and V_{DDL}. The PMU consists of a voltage selector and two single pole-single throw (SPST) switches. The voltage selector multiplexer is implemented with a single pole-double throw (SPDT) switch that is controlled by the voltage on node Y as shown in Fig. 5.9. The voltage selector supplies the MCU with V_{DDH} for a logic high input and V_{DDL} for a logic low input. The

Hypnos implementation utilizes a network of normally-open (NO) and normally-closed (NC) SPST switches that provide isolation between different modules. The SPDT and SPST switches are chosen such that the power overhead is minimized. The NO/NC SPST switches (MAX4645/6) that constitute the PMU and interrupt interface is chosen for its ultra low current consumption of 100 pA [93] each. For the SPDT voltage selector multiplexer, we choose the ADG819 [94].

Active to $\mathbf{LPM_H}$ transition

As discussed in Section 5.3, the PMU is controlled by a GPIO, `PCTRL`. Fig. 5.9 shows the two paths that are controlled by `PCTRL`. When the MCU is in active mode, it sets `PCTRL` to logic high. This has two implications, the first of which is the isolation of node Y from the interrupt interface module. Switch $S2$, which connects the nodes X and Y is of NC type, and therefore when it receives a logic high, the connection is severed. Secondly, `PCTRL` activates *path 1* (shown in red), which pulls node Y to a logic high and consequently reinforces the selection of V_{DDH}.

As discussed in Section 5.3.2, transitioning to LPM_H from the active mode is preceded by configuring the interrupt interface and pulling `PCTRL` to logic low. The reset latch signal (`RSTL`) is used for this purpose and in particular, to configure the interrupt interface. Setting `RSTL` to logic high configures the interrupt interface by pulling node X to ground and thereby, closing $S3$. Simultaneously, `PCTRL` is set to logic low for opening $S1$ and closing $S2$, which activates *path 2* (shown in green). As a result, the MCU supply voltage drops to V_{DDL} and it enters LPM_H. The wake up operation of the PMU is discussed in the following subsection.

5.5.3 Interrupt interface

The primary functionality of the interrupt interface is to preserve the interrupt signal until the microcontroller wakes up and is ready to register the interrupt. In our system, the MCU is configured to register a positive edge-triggered interrupt on $P2$.

The interrupt interface consists of a unidirectional latch that captures a low-to-high transition. The working of the unidirectional latch and the MCU wake-up is described as follows.

Wake-up from $\mathbf{LPM_H}$

On receiving an interrupt signal, the switch $S7$ closes, pulling `INT` and node X to logic high. This opens $S3$ and node Y gets charged to V_{DDH} causing the PMU to supply the microcontroller with V_{DDH}. Simultaneously, $S5$ connects the logic high voltage to node Z, which closes $S4$. In the event that the wake-up overhead of the MCU is more than the pulse width of the interrupt signal, it is important to latch the interrupt. The positive feedback from node X through node Z helps node X to retain V_{DDH} via $S4$.

Once the MCU wakes up, `PCTRL` is set, which disconnects the interrupt interface from the PMU by opening $S2$. Then, once the interrupt is registered by the MCU, it is reset using `RSTL`. When `RSTL` is logic high, it closes $S6$ and pulls node Z to ground, and voltage at node X discharges until $S3$ closes, upon which node X is immediately pulled to the ground voltage. Thus, the interrupt interface is ready to latch the next interrupt.

5.6 Experimental results

In this section, we describe the experimental setup, the application used for evaluation, and the baselines with which we compare Hypnos. Finally, we present the results obtained and discuss the trade-offs involved.

5.6.1 Experimental setup

For our experiments, a Tektronix 6430 Keithley source meter is used as the power supply. It can act as a voltage source as well as measure currents as small as fem-

toamperes. All power measurements are made using the source meter itself. The V_{DDH} for all the experiments is set to 2.2 V. Lastly, all the latency overheads are recorded using a Tektronix MDO4104-3 oscilloscope.

Evaluation application

For evaluation, we consider a simple periodic sensing application. The application is described as follows. The microcontroller wakes up on receiving an external interrupt, and then collects a few samples following which an averaging operation is performed. As soon as the computations are completed, the application is put into the idle state. After a few such sense-sleep cycles, the application transmits the gathered data.

Baselines

We use two baselines, which are described below, to compare the energy and latency benefits of our proposed technique, Hypnos.

Conventional LPM The first baseline that we choose to compare with is the conventional LPM (*CLPM*) that is currently prevalent in microcontrollers and corresponds to the *shallow sleep* mode. In particular, the LPM4 mode of the MSP430G2452 is chosen as *CLPM*, as it is the least power consuming data retention mode available. In *CLPM*, power is supplied to the internal registers and SRAM to retain the state. Power consumption of the MCU is reduced by halting the clock and shutting off the peripherals. Typically, the MCU wakes up from LPM4 through a `WInt`. Note that *CLPM* does not require the additional peripheral circuitry that is needed for implementing Hypnos. The MSP-TS430PW28A evaluation board was used for evaluating *CLPM*.

Checkpointing to non-volatile flash The second baseline that we compare HYPNOS with is a flash-based checkpointing scheme (*NVLPM*). We denote the low power mode for the *NVLPM* scheme as LPM_{NV}. *NVLPM* corresponds to the *deep sleep* mode, which is the lowest power consuming mode available in advanced MCUs of today, wherein the SRAM and register files are also powered off. In LPM_{NV}, the MCU consumes power to keep the GPIOs related to the `WInt` active. Typically, this power ranges in the order of tens of nanoamperes. For applications with a very low duty cycle, the RAM data, processor register contents, and peripheral configurations may be checkpointed into the available on-chip flash memory to retain the state. Once the checkpointing is completed, the system goes into a low power mode wherein it waits for a `WInt`. Unfortunately, the MSP430G2452 MCU does not possess such a mode. Therefore, we create an LPM_{NV} mode that has an interface similar to HYPNOS, which powers off the MCU when it enters LPM_{NV}. Thus, the implementation for *NVLPM* is same as that of HYPNOS except that $V_{DDL} = 0$ V. Note that *NVLPM* implemented this way is more energy-efficient than conventional LPM_{NV}s as the GPIOs are completely powered off. *NVLPM* has an additional power overhead component as compared to HYPNOS due to the periodic erase operations required by flash. The MSP430G2452 MCU has 8 kB of flash memory divided into 16 segments of 512 B each. A segment is the smallest unit of memory that can be erased. Hence, for a checkpoint size of 256 B, an erase operation has to be performed once in every two cycles (in steady state).

5.6.2 Latency overhead

The latency overhead associated with HYPNOS arises from the time taken by the wake-up, recall, and sleep steps. We define the wake-up overhead to be the time taken by the processor to be able to begin code execution from the time the MCU receives the interrupt. In particular, it incorporates the time taken by the PMU to toggle the supply voltage from V_{DDL} to V_{DDH}. The sleep overhead is the time taken for the

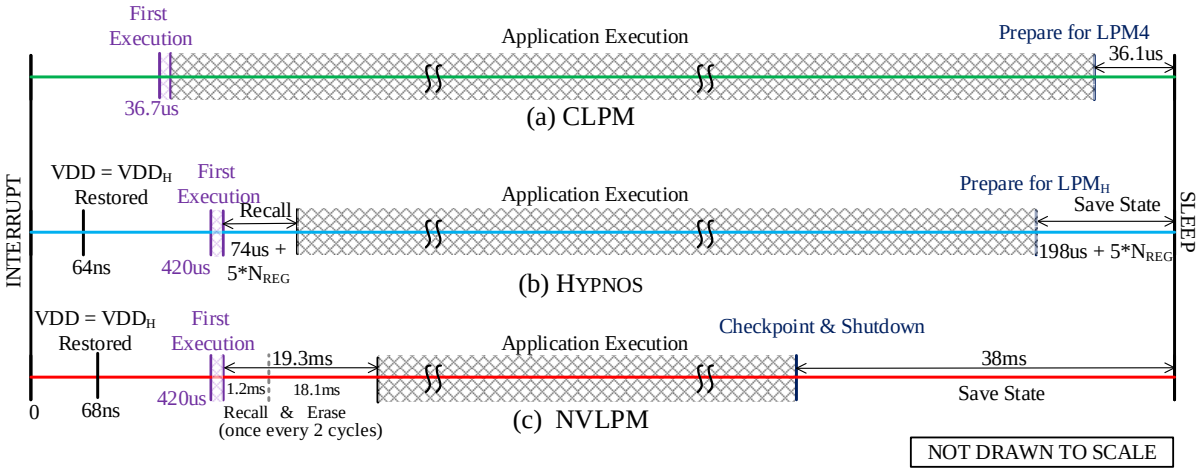


Fig. 5.11. Latency overhead comparison

Table 5.2.
Time-interval definitions

t_{wakeup}	Time from interrupt to first execution.
t_{recall}	Time required to recall the previous state
t_{sleep}	Time required to save state and enter sleep from the time of issuing command

system to enter LPM_H from the time `enter_lpm()` is invoked, and includes the time taken for checkpointing. Recall overhead is the overhead associated with restoring the checkpointed state. The latency overhead for the three schemes is shown in Fig. 5.11 and Table 5.2 defines the time intervals. Among the three schemes, *CLPM* is a complete *in-situ* retention solution, *NVLPM* is a complete checkpointing solution, while HYPNOS is a mixture of both.

CLPM has the least t_{wakeup} because the MCU V_DD is V_DDH in LPM4. On the other hand, HYPNOS and *NVLPM* have similar and larger t_{wakeup} due to the lower V_DDL voltage being used in the respective low power modes. Both HYPNOS and *NVLPM* undergo a brown-out reset (BOR) on waking up in addition to stabilizing internal

PLLs, clocks, *etc.* The time required for the MCU V_{DD} to reach V_{DDH} is measured as 64 ns for HYPNOS and 68 ns for *NVLPM*. The slight variation in charging times for HYPNOS and *NVLPM* is due to the difference in their respective V_{DDL} voltages. The charging is quick because of the placement of C_{decap} on the V_{DDH} rail before the voltage selector multiplexer. Thus, only the effective capacitance of the MCU needs to be charged. After waking up, the microcontroller initializes its peripherals. This is shown in the figure as *First Execution*. The time required for initializing is application dependent and is same for all the three schemes. Hence, its overhead is not separately accounted for in our calculations.

The length of t_{recall} is determined by the number of registers that need to be checkpointed. HYPNOS requires a t_{recall} of 74 μs to restore the 30 B of checkpointed data pertaining to the processor registers. Any additional GPIO register that needs to be checkpointed incurs an additional latency of 5 μs. On the contrary, *NVLPM* has a t_{recall} of 1.2 ms arising from the larger data transfer overhead due to a checkpoint size of 256 B that includes SRAM contents in addition to the processor registers. Similar to LPM_H, checkpointing each additional peripheral register incurs 5 μs overhead. Further, for *NVLPM*, an already written flash-segment has to be erased before it can be written to again. The erase operation adds a latency of $t_{erase} = 18.1$ ms to *NVLPM*'s recall overhead once every two cycles. Finally, the t_{sleep} of the three solutions are compared. The t_{sleep} of *CLPM* encompasses two operations. First, *CLPM* configures the directions of GPIOs and associated pull-up resistors to appropriate values in order to reduce the leakage power consumption at the GPIOs. Then, the `WInt` needs to be enabled before issuing the command to shut-off the clocks and transition into LPM4. HYPNOS' t_{sleep} encompasses the overhead for *CLPM* and also includes the checkpointing overhead incurred in retaining processor registers, GPIO registers, and in executing the mandatory operations discussed in Section 5.3.2. On the contrary, *NVLPM* has a large t_{sleep} as it utilizes flash memory to store data. A flash write operation is as cumbersome as the flash erase operation and as a result, dominates the t_{sleep} duration.

Table 5.3.
Idle mode current consumption

	Current consumption (nA)
$\text{LPM}_{\text{H}V}$	26
$\text{LPM}_{\text{H}E}$	1
LPM_{NV}	1
Conventional LPM	102

Active mode current consumption for the microcontroller = 296 μA

5.6.3 Power consumption

Table 5.3 shows the idle mode current consumption of HYPNOS, *CLPM*, and *NVLPM*. For differentiating between the two different V_{DDL} generation techniques and their corresponding low power mode, we use the notations $\text{LPM}_{\text{H}V}$ and $\text{LPM}_{\text{H}E}$. The notation $\text{LPM}_{\text{H}V}$ corresponds to the case when V_{DDL} is derived from V_{DDH} and the notation $\text{LPM}_{\text{H}E}$ corresponds to the resultant low power mode where an energy harvesting source is used to generate V_{DDL}. When the MCU is in $\text{LPM}_{\text{H}V}$, the active system components include all the switches, the voltage converter, and the voltage selector. A V_{DDL} of 220 mV is achieved for HYPNOS using a series resistor as explained in Section 5.5, which makes the total power consumption for HYPNOS in idle mode to be 57.2 nW. *CLPM* does not have any additional circuits, and only the microcontroller contributes to the power consumption. Hence, the power consumption for LPM4 is 224.4 nW. Thus, $\text{LPM}_{\text{H}V}$ decreases the idle mode power consumption by `4x` as compared to a conventional implementation. As explained before, the MCU is completely turned off in LPM_{NV}. Hence, the current consumed in LPM_{NV} is due to the peripheral circuitry making up the HYPNOS hardware architecture, which is measured to be 1 nA. In other words, out of the 26 nA consumed for $\text{LPM}_{\text{H}V}$, only

1 nA accounts for the peripheral circuitry. These additional components contribute only 2.2 nW of the 57.2 nW of power that is consumed in $\mathrm{LPM}_{\mathrm{H}V}$. Hence, the major portion of the consumed power is spent on generating the required voltage of $\mathrm{V}_{\mathrm{DRV}}$ through the use of the resistive divider. When $\mathrm{V}_{\mathrm{DRV}}$ is generated by energy harvesting, the power cost of implementing the low power mode is further reduced to that of powering just the peripheral circuitry, which is just 2.2 nW. Thus $\mathrm{LPM}_{\mathrm{H}E}$ consumes 100x lower power than LPM4.

In order to quantify the energy consumption of HYPNOS, we devise an energy model as given by the following equations. The latency overhead presented by each approach is represented by T_{OH} as shown in Equation (5.1). In addition, T_1 refers to the duration spent by the MCU in executing useful application tasks, and T_2 indicates the time spent in sleep mode. Equation (5.6) defines $T_1^{'}$ as the total time for which the MCU is not in sleep mode. We use $T_1^{'}$ to compute the duty cycle (Equation (5.7)) for a particular T_1 and T_2.

$$T_{OH,i} = t_{wakeup,i} + t_{recall,i} + t_{sleep,i} \tag{5.1}$$

$$i \in \{\text{HYPNOS}, CLPM, NVLPM\}$$

$$E_{\text{HYPNOS},V} = P_{Active} * (T_1 + T_{OH,\text{HYPNOS}}) + P_{LPM_{\mathrm{H}V}} * T_2 \tag{5.2}$$

$$E_{\text{HYPNOS},E} = P_{Active} * (T_1 + T_{OH,\text{HYPNOS}}) + P_{LPM_{\mathrm{H}E}} * T_2 \tag{5.3}$$

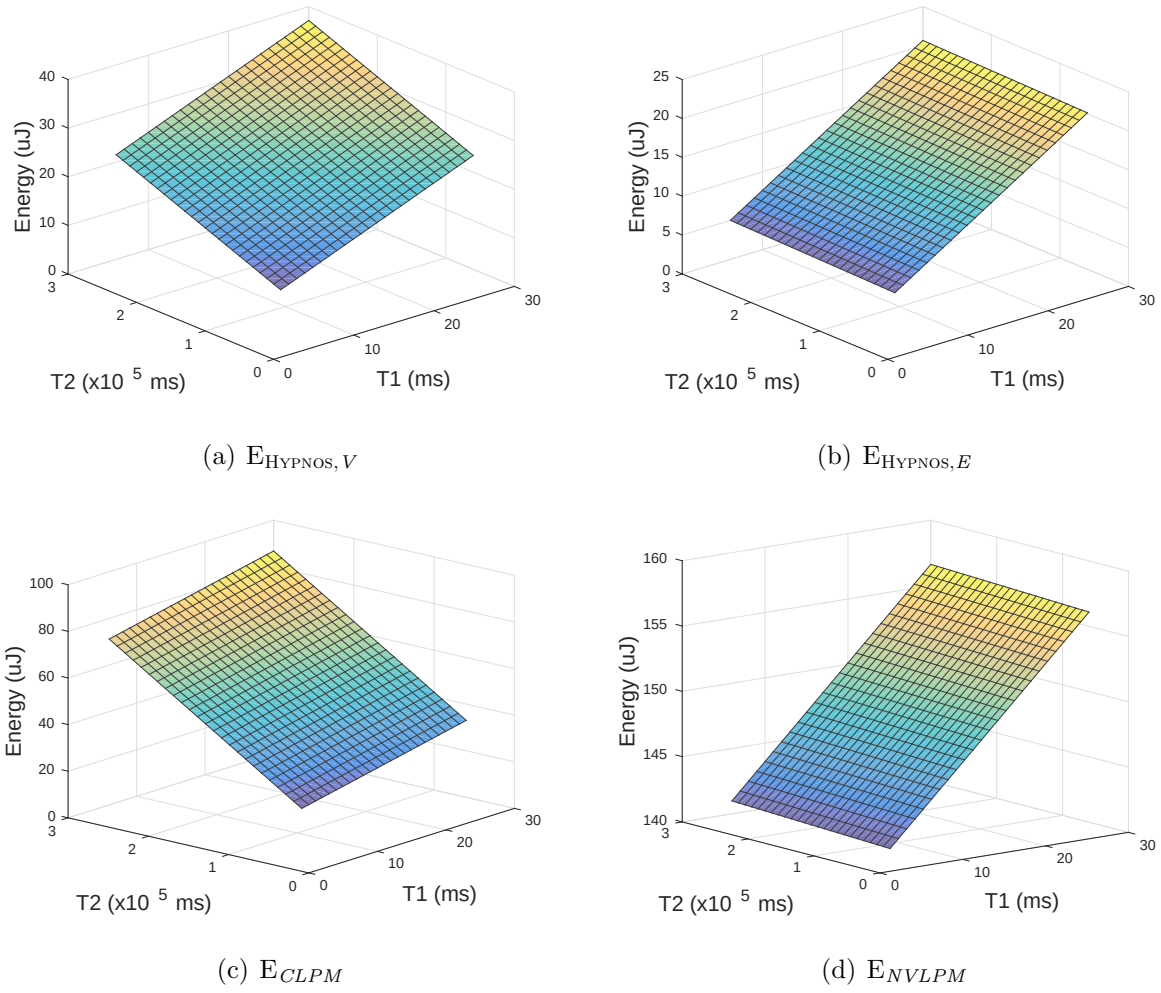
$$E_{CLPM} = P_{Active} * (T_1 + T_{OH,CLPM}) + P_{LPM4} * T_2 \tag{5.4}$$

$$E_{NVLPM} = P_{Active} * (T_1 + T_{OH,NVLPM} - t_{sleep,nvlpm}) + P_{write} * t_{sleep,nvlpm} + \frac{P_{erase} * t_{erase}}{2} + P_{LPM_{NV}} * T_2 \tag{5.5}$$

$$T_1^{'} = T_1 + T_{OH} \tag{5.6}$$

$$Duty\ Cycle(\%) = \frac{T_1^{'}}{T_1^{'} + T_2} * 100 \tag{5.7}$$

The current consumption during $T_1^{'}$ was measured and found out to be close to the active mode current consumption. While switching to active mode, an instantaneous surge in current consumption is noticed. This inrush current is observed to be 296 μA.



(a) $E_{HYPNOS,V}$

(b) $E_{HYPNOS,E}$

(c) E_{CLPM}

(d) E_{NVLPM}

Fig. 5.12. Energy consumption for the MCU across different active and idle durations

As we are not privy to the internal architecture of the MSP430G2452 MCU, we can only speculate about the reason for the high inrush current. We presume that when the MCU enters low power mode, the internal circuitry switches off a few logic paths and power-gates certain modules as a power saving strategy. Hence, the inrush current may be due to the sudden additional capacitance load presented to the supply upon wakeup. Finally, P_{erase} and P_{write} correspond to the power required to erase and write to the flash memory. The measured values for the same are 5.72 mW and 2.2 mW respectively. Equating equation (5.5) with equations (5.2) and (5.2)

shows that duration of sleep required for *NVLPM* to break-even in terms of energy consumption is 40 minutes for $\mathrm{LPM_{H}}_V$ and over 17 hours for $\mathrm{LPM_{H}}_E$.

Using this model, we depict the average energy consumption (Eqs. (5.2) - (5.5)) by varying T_1 and T_2 in Fig. 5.12[3]. We consider any heavily duty cycled application for our experiments. Therefore, T_1 is varied from 6 ms to 30 ms, while T_2 is varied from 60 s to 300 s. Fig. 5.12 shows that in spite of having almost zero idle-time power consumption, E_{NVLPM} is much more than E_{CLPM} and $E_{\mathrm{HYPNOS}(V,E)}$. This can be attributed to the large power demand and huge latency overhead of the flash erase and write operations that dominate E_{NVLPM}. The dependence of E_{NVLPM} on T_2 is negligible as compared to T_1.

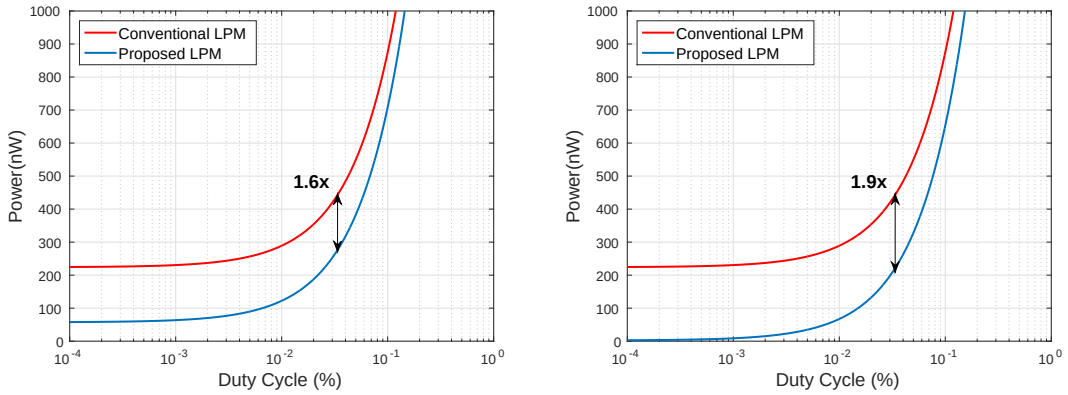
The advantage of $\mathrm{LPM_H}$[4], an ultra low power idle mode over a conventional idle mode implementation is evident by comparing Figs. 5.12(a) and 5.12(b) with Fig. 5.12(c). To begin with, the improvement in power savings due to $\mathrm{LPM_H}$ translates to overall energy savings for a system operating with a low duty cycle. For any T_1, $\mathrm{LPM_H}$ consumes lesser energy than *CLPM* due to the lower slope of $E_{\mathrm{HYPNOS}(V,E)}$ with respect to T_2. Further, the impact of the energy consumed due to the additional latency overhead of $\mathrm{LPM_H}$ becomes insignificant in heavily duty cycled systems. The slope of $E_{\mathrm{HYPNOS}(V,E)}$ and E_{CLPM} with respect to T_1 is a testament to the dominating effect that idle time power consumption places on the overall energy cost. In other words, this signifies that the energy cost due to the higher latency overhead for $\mathrm{LPM_H}$ gets amortized in heavily duty cycled systems.

The average system power consumption when utilizing LPM4 or $\mathrm{LPM_H}$ as the low power mode is given by the following equation where $\mathrm{P_{idle}}$ denotes the respective sleep-mode's power consumption.

$$P_{avg} = \frac{P_{Active} * T_1^{'} + P_{idle} * T_2}{T_1^{'} + T_2} \tag{5.8}$$

[3]For this model, we assume that no GPIO registers are saved

[4]In this section, we refer to both $\mathrm{LPM_{H}}_V$ and $\mathrm{LPM_{H}}_E$ as $\mathrm{LPM_H}$. Whenever distinction is required, the appropriate notation is used.



(a) 1.6x reduction in power consumption in $\mathrm{LPM}_{\mathrm{H}V}$

(b) 1.9x reduction in power consumption in the proposed $\mathrm{LPM}_{\mathrm{H}E}$

Fig. 5.13. Average power vs duty cycle for MCUs utilizing different sleep-mode schemes

Figs. 5.13(a) and 5.13(b) compares the average power consumption of HYPNOS and *CLPM* schemes as a function of the operational duty cycle. Lower duty cycles translate to larger power savings for HYPNOS as a direct result of spending more time in the idle state. For example, consider a typical sensing system that collects samples for a duration of 100 ms ($T_1' = 100$ ms) once every 5 minutes ($T_2 = 299.9$ s). Then, the microcontroller of that system, which has a duty cycle of $3.33 * 10^{-2}\%$, is `1.6x` power efficient than LPM4 when implementing $\mathrm{LPM}_{\mathrm{H}V}$ and `1.9x` power efficient when utilizing $\mathrm{LPM}_{\mathrm{H}E}$. For lower duty cycles, the disparity in average power consumption between the conventional low power mode and $\mathrm{LPM}_{\mathrm{H}}$ increases. However, for systems with higher duty cycles, the power consumption gets dominated by the active mode power. Therefore, the power consumption for both HYPNOS and *CLPM* schemes become comparable and for applications that have duty cycle greater than 0.5%, the power difference is almost negligible.

Impact of V_{DDL} generation by energy harvesting on power consumption

As can be inferred from the above discussion, $P_{LPM_{NV}}$ corresponds to the power consumed by the HYPNOS peripheral circuitry. Hence, when V_{DDL} is generated utilizing the photodiode, the power consumption in LPM_{HE} is equal to that in LPM_{NV}, *i.e.*, $P_{LPM_{HE}} = P_{LPM_{NV}}$. Therefore, putting the MCU into LPM_{HE} is more energy-efficient than putting it into LPM_{NV} for any duration of the sleep-mode, T_2, as the cost of entering and exiting LPM_{NV} is much larger as compared to the overhead for entering and exiting LPM_{HE}. On the other hand, $E_{CLPM} \geq E_{HYPNOS}$ for $T_2 > 3$ s and for any T_1. Therefore, when the idle mode duration exceeds 3 s, LPM_{HE} is energy efficient than *CLPM*. Hence HYPNOS, which provides an additional V_{DDL} rail for MCUs, enables a new low power mode that is `102x` times (from Table 5.3) better than *CLPM* in terms of current consumption. Finally, Fig. 5.12(b) shows the energy consumption of HYPNOS while harvesting from the photodiode, and Fig. 5.13(b) compares the power consumption of *CLPM* with HYPNOS when V_{DDL} is generated through energy harvesting. As compared to Fig. 5.12(a), the energy consumption for LPM_{HE} is reduced when utilizing the energy output from the photodiode. Similarly, decrease in idle mode power consumption results in a larger gap in the average power consumed between LPM4 and LPM_{HE} for applications with low duty cycle as is shown in Fig. 5.13(b).

5.7 Discussions

In this section, we discuss the impact that the HYPNOS scheme has on the overall energy consumption of a system using an example application followed by a discussion on the complexity of implementing HYPNOS in an MCU.

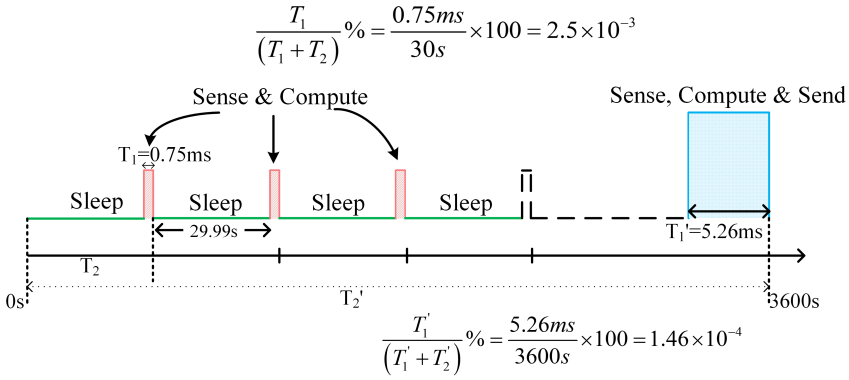


Fig. 5.14. Periodic sense & send application

5.7.1 Impact on application-level energy consumption

For heavily duty-cycled embedded applications, a reduction in sleep mode power consumption translates to significant savings in application-level energy consumption. In order to quantify the far-reaching impact that HYPNOS has on the overall energy consumption, we consider the wireless embedded application described in Section 5.6.1. Assume that the system performs a sensing operation every 30 seconds, wherein three samples are acquired and their average is computed. These samples are then stored in the RAM before the system enters sleep mode. This cycle is repeated 120 times and the gathered data is transmitted by the system once every hour. Fig. 5.14 shows the application behavior and two different duty cycles. The first is the duty cycle for sampling, and the second is the duty cycle for radio transmissions. The red rectangles correspond to the sensor measurements and accompanying computations. The width of the rectangle represents the active time while the height roughly denotes the power consumption. The much larger blue rectangle depicts the power consumption and time required to perform the radio transmission in addition to a sense and compute operation. Table 5.4 documents the cumulative time spent by the application in each task for a time-span of one hour. The corresponding energy consumption per task for *CLPM* and HYPNOS, using both the resistive divider and the photodiode for V_{DDL} generation, are shown. The components considered for the

Table 5.4.
Break-up of overall energy consumption for different sleep-mode schemes

Application Task	Cumulative Time Spent per hr (ms)	Energy per hr *CLPM* (μJ)	Energy per hr HYPNOS (μJ)	Energy per hr HYPNOS using energy harvesting (μJ)
Sense[a] & Compute[a]	89.14	26.2	64.88	64.88
Sleep[a]	$35.9 * 10^5$	807.83	205.91	7.92
Sense, Compute & Send[b]	5.26	66.61	66.94	66.94
Total Energy per hour (μJ)		900.64	337.73	139.74
Reduction in overall energy consumption		1	2.67x	6.45x

[a]Measured from experiment [b]Computed from datasheet

system are the MSP430G2452 MCU, the TMP20 analog temperature sensor from Texas Instruments, and the AT86RF233 2.4 GHz RF transceiver from Atmel.

As is evident from Table 5.4, the sleep mode energy dominates the overall energy consumption. $\text{LPM}_{\text{H}E}$ accounts for only 5.7% of the total energy consumption when the HYPNOS scheme is employed, whereas LPM4 (in *CLPM*) accounts for 89.7% of the total energy consumption. Note that $\text{LPM}_{\text{H}E}$ is over `100x` energy efficient than LPM4. However, the HYPNOS scheme consumes more energy (`1.42x`) than *CLPM* for both sensing and sending operations due to the additional overhead present in waking up and sleeping. In spite of this, the proposed HYPNOS scheme achieves a reduction in application level energy consumption of `6.45x` over the *CLPM* scheme.

Choosing V_{DRV} and its impact on energy consumption

For any MCU, deciding the value of V_{DDL} for HYPNOS involves a characterization step to find its V_{DRV}. As seen in Section 5.2.2, the V_{DRV} for the MSP430G2452 MCU was chosen to be 220 mV as it was the highest DRV seen in the characterization experiment across 20 different chips. However, since it is infeasible to perform exhaustive characterization, process variations could still result in a specific MCU having a data retention voltage that is higher than the V_{DRV} found by characterization. Hence, as mentioned in Section 5.2.3, we propose guard-banding the V_{DRV} as a solution to this issue.

Guard-banding the V_{DRV} of an MCU will result in a higher sleep-mode power consumption when using a voltage converter, or increase the minimum light intensity required for data retention while supplying power using the photodiode output. For example, setting a guard-banded V_{DRV} of 320 mV for the MSP430G2452 MCU results in an increased power consumption of 112 nW in LPM_{HV}, or places a minimum requirement of 36 lux for supporting LPM_{HE} with the photodiode. The first scenario results in a degradation in the energy-reduction achieved by HYPNOS. For the application scenario illustrated in Fig. 5.14, the guard-band results in an increase of sleep-mode energy consumption from 205.91 μJ to 403.9 μJ, thus lowering the system-level energy reduction provided by HYPNOS from 2.67x to 1.68x. However, when using energy harvesting as the power supply during LPM_{HE}, there is no degradation in the energy reduction provided by HYPNOS and it remains the same at 6.45x, albeit with a higher constraint on the minimum light intensity required to support LPM_{HE}.

5.7.2 LPM_H Implementation in an MCU

For implementing HYPNOS in an MCU, two critical requirements need to be satisfied. First, the additional voltage (V_{DDL}) has to be generated with minimum power overhead. The above discussions elucidate the fact that employing an energy harvest-

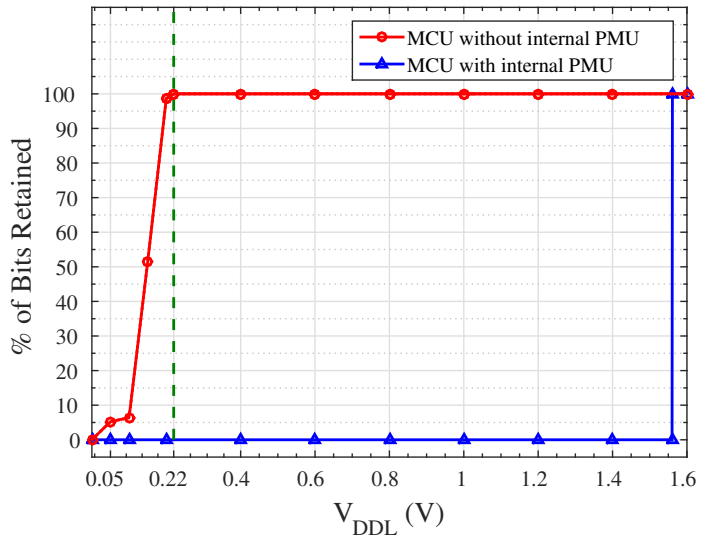


Fig. 5.15. Comparison of SRAM data retention with varying V_{DDL} for MCUs with and without an internal PMU

ing source could enable a full retention low power mode, similar to a *shallow sleep* mode, with ultra-low power consumption comparable to that of a *deep sleep* mode. Note that this is made possible by using just a photodiode that occupies minimal board-area (2.8 mm×2.4 mm), and which is already present as a light sensor in many embedded systems. However, the MSP430G2452 MCU used in this work does not have a complex internal PMU. Therefore, we repeat the experiment conducted in Section 5.2.2 with the TI MSP430F5438A MCU, which has an internal PMU that consists of a supply voltage supervisor (SVS), a low dropout regulator (LDO), and a reset circuitry. The PMU generates and provides the voltages required by the processing core, on-chip flash, on-chip SRAM, peripherals, *etc.*, from the supply voltage. Fig. 5.15 shows the impact of an internal PMU on the MCU's capability to retain data as the supply voltage is scaled. Of the many voltage domains that are present in the microcontroller, the voltage domain corresponding to the processing core and on-chip SRAM is the one supplied by the V_{CORE} rail[5]. We find that when V_{DDL} is lowered be-

[5]In the MSP430F5438A MCU, the V_{CORE} rail is brought out as a GPIO pin. In our experiment, we make voltage measurements using the same.

yond 1.56 V (say V_{OFF}), the V_{CORE} rail is switched off by the PMU. Hence, the power supply to the internal SRAM is cut off completely making data retention infeasible for voltages lesser than V_{OFF}. The power management unit of most microcontrollers of today do not allow for the direct control of the SRAM voltage below the specified range of normal operation. If the PMU design in these microcontrollers could be modified such that the power supply to the internal SRAM be brought out to an external pin, then energy harvesting could be utilized to generate the V_{DRV} required to retain the data in SRAM. Such a design would allow the SRAM to be powered by the energy harvesting source only when it is in retention mode while utilizing the voltages generated by the internal PMU in active mode. Second, the relevant GPIOs that should register a `WInt` should be powered on. In existing designs, the GPIOs receive adequate power to register a wakeup interrupt and a similar approach would guarantee a wakeup from LPM_H. This could possibly result in additional power reduction from the current Hypnos implementation, as the amount of accompanying logic required to support n `WInts` would be significantly reduced. Most importantly, the entire interrupt interface can be eliminated, although the power selection unit should be replicated at the IO level of the MCU.

5.8 Related work

SRAM retention and the associated data retention/ hold voltage has been a topic of sufficient interest and research in the past as mentioned in Section 5.2.1. Conventionally in *shallow sleep* modes of MCUs, registers and other related circuitry in addition to the SRAM memory are kept powered on. On the contrary, Hypnos keeps just the SRAM powered on at V_{DRV} by scaling the supply voltage of the MCU during idle time.

Over the past decade, there has been sufficient interest in exploring data retention and remanence characteristics of different memories. This has primarily been driven by the need for understanding and characterizing potential security vulnerabilities

of systems. Refs. [95, 96] explore data remanence of memories in the context of cold boot system attacks. Ref. [97] explores SRAM data remanence for generating random numbers in RFID tags.

The variation of SRAM data retention with changes in power-supply voltage has also been explored by Ref. [98]. The work aims to quantify the different retention times associated with different commercial off-the-shelf SRAM chips to prove the existence of data remanence and the negative impact that remanence has on system security. Recent work has proposed the use of an MCU's chip V_{DRV} characteristic as a fingerprint for chip identification [99]. The authors also observe the loss of processor state when the core voltage is lowered and perform checkpointing of state to a non-volatile memory. In contrast, HYPNOS reduces power consumption in idle mode by lowering the core voltage and stores state information *in-situ* in the SRAM itself. Tardis [100] is an algorithm designed to utilize SRAM data decay with loss in supply voltage to provide a notion of elapsed time. Tardis quantifies the amount of data decay, which is characterized by the number of bit flips that occur due to a loss of power, and use it to thwart repeated security attacks within a short-span of time. On the other hand, HYPNOS defines a HW/SW architecture that ensures no bit flips occur while going to an ultra-low power data retention idle mode. Idealvolting [101] reduces the voltage beyond the manufacturer-specified operating voltage in active mode. Reducing the SRAM supply voltage below V_{DRV} has recently been explored for approximate computing [102]. Finally, this work has been presented in Refs. [103,104].

5.9 Summary of contributions

In this chapter, we have proposed and implemented HYPNOS, a hardware-software architecture that reduces the overall energy consumption in heavily duty cycled applications. This is achieved by introducing a novel low power mode, LPM_H that retains data in the on-chip SRAM of embedded microcontrollers (which have the processing core and SRAM share the same voltage rail) by performing extreme voltage scaling

of the MCU's supply voltage. We demonstrate that our proposed LPM_H consumes only 1 nA when powered using a light sensing photodiode, which is over `100x` better than existing LPMs. Using an MSP430G2452 MCU, we showed that putting it into LPM_H is more energy-efficient than a conventional shallow sleep mode if the idle time is more than 3 s. For an example application, we quantified the reduction in the overall energy consumption to be `6.45x` when utilizing LPM_H as opposed to *CLPM*. Thus by performing extreme voltage scaling during idle time, our proposed HYPNOS architecture defines a new low power mode (LPM_H) whose power consumption is comparable to a *deep sleep* mode while still being able to retain data like a *shallow sleep* mode.

6. SUMMARY

The Internet of Things (IoT) is poised to pervade all facets of human life with the vision of improving everyday life such that human energy could be diverted to perform and solve more attractive problems. The variety of societal-scale problems that the IoT seeks to tackle include telemetry, healthcare, home automation, energy conservation, security, wearable computing, asset tracking, maintenance of public infrastructure, waste management, environmental monitoring, and so on. The work-horses of IoT are the devices that sense and communicate different physical phenomena of interest to the cloud. The cloud then processes the data and makes intelligent decisions upon it. Various forecasts estimate that around 50 billion devices will be deployed by 2020. Powering such a large amount of devices is challenging due to various factors such as the need for untethered operation, adverse deployment location, stringent form-factor constraints, *etc.* While a battery-based approach is highly enticing, the cost and effort of performing maintenance for billions of devices renders it infeasible. Energy harvesting has long been thought of as a promising solution, but the unreliable and intermittent nature of ambient sources has deterred designers from adopting it in the past. An emerging class of embedded devices called intermittently-powered IoT devices, envisions to work on scanty and unreliable ambient energy sources. However, performing computations energy-efficiently and reliably in the face of frequent and sudden power loss remained a challenging proposition.

In the first part of this dissertation (Chapter 3), we proposed a solution for intermittently-powered IoT devices to perform computations reliably and energy-efficiently. We made a case for an emerging non-volatile memory, Ferroelectric RAM to be used as unified memory to enable *in-situ* checkpointing such that more energy per power cycle could be used for executing computations as compared to a conventional memory architecture using Flash. *In-situ* checkpointing does away with the

data transfer overheads required for storing the state and hence, reduces the check-pointing energy overhead. The chapter contributions included a proposed software flow for these systems to enable application execution in a seamless and transparent manner.

In the second part of the dissertation, we built upon the contributions made in Chapter 3 to further enhance energy efficiency and performance of these systems (Chapter 4). This is achieved by a judiciously mapping different program sections across SRAM and FeRAM, such that the beneficial characteristics of both kinds of memories could be utilized. We proposed a run-time dynamic memory mapping scheme that finds an energy-optimal memory mapping at the granularity of functions that constitute a program. We also proposed a technique called *Energy-Align*, that performs proactive system shutdown to further improve the energy-efficiency and performance of the system.

The third part of the dissertation (Chapter 5) addressed IoT devices that operate intermittently and reduced the sleep-mode energy consumption of these devices. We increased the lifetime of the battery by reducing the sleep mode power consumption to such an extent that the sleep mode could be sustained by energy harvesting alone even in the harshest of conditions. This was achieved by intelligently exploiting SRAM's device characteristics at the system-level, by performing sleep mode voltage scaling of the MCU's supply voltage.

In summary, this dissertation makes a significant step in enabling IoT devices to achieve a *set-and-forget* mode of operation energy-efficiently and reliably. We hope that the solutions presented in this dissertation will facilitate in the widespread adoption of IoT devices and help in realizing the vision of the Internet of Things.

REFERENCES

REFERENCES

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything," 2011. [Online]. Available: http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, "Powering the Internet of Things," *Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on*, pp. 375–380, Aug 2014.

[3] H. Jayakumar, A. Raha, Y. Kim, S. Sutar, W. S. Lee, and V. Raghunathan, "Energy-efficient system design for IoT devices," *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 298–301, Jan 2016.

[4] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," *ESSCIRC (ESSCIRC), 2012 Proceedings of the*, pp. 149–152, 2012.

[5] S. Bartling, S. Khanna, M. Clinton, S. Summerfelt, J. Rodriguez, and H. McAdams, "An 8MHz 75μA/MHz zero-leakage non-volatile logic-based Cortex-M0 MCU SoC exhibiting 100% digital state retention at VDD=0V with <400ns wakeup and sleep transitions," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pp. 432–433, 2013.

[6] S. Khanna, S. C. Bartling, M. Clinton, S. Summerfelt, J. A. Rodriguez, and H. P. McAdams, "An FRAM-Based Nonvolatile Logic MCU SoC Exhibiting 100% Digital State Retention at VDD= 0 V Achieving Zero Leakage With < 400-ns Wakeup Time for ULP Applications," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 95–106, Jan 2014.

[7] A. Baumann, M. Jung, K. Huber, M. Arnold, C. Sichert, S. Schauer, and R. Brederlow, "A MCU platform with embedded FRAM achieving 350nA current consumption in real-time clock mode with full state retention and 6.5 μs system wakeup time," *VLSI Circuits (VLSIC), 2013 Symposium on*, pp. C202–C203, 2013.

[8] V. Singhal, V. Menezes, S. Chakravarthy, and M. Mehendale, "8.3 A 10.5 μA/MHz at 16MHz single-cycle non-volatile memory access microcontroller with full state retention at 108nA in a 90nm process," *Solid- State Circuits Conference - (ISSCC), 2015 IEEE International*, pp. 1–3, Feb 2015.

[9] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. Eversmann, "An 82 μA/MHz microcontroller with embedded FeRAM for energy-harvesting applications," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pp. 334–336, 2011.

[10] T. Instruments, "Msp430fr573x datasheet," April 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430fr5739.pdf

[11] N. Sakimura, Y. Tsuji, R. Nebashi, H. Honjo, A. Morioka, K. Ishihara, K. Kinoshita, S. Fukami, S. Miura, N. Kasai, T. Endoh, H. Ohno, T. Hanyu, and T. Sugibayashi, "10.5 A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 184–185, Feb 2014.

[12] Panasonic, "Mn101lr05d/04d/03d/02d datasheet."

[13] Y. Liu, Z. Wang, A. Lee, F. Su, C. P. Lo, Z. Yuan, C. C. Lin, Q. Wei, Y. Wang, Y. C. King, C. J. Lin, P. Khalili, K. L. Wang, M. F. Chang, and H. Yang, "4.7 A 65nm ReRAM-enabled nonvolatile processor with 6x reduction in restore time and 4x higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic," *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 84–86, Jan 2016.

[14] A. Sheikholeslami and P. Gulak, "A survey of circuit innovations in ferroelectric random-access memories," *Proceedings of the IEEE*, vol. 88, no. 5, pp. 667–689, 2000.

[15] G. R. Fox, F. Chu, and T. Davenport, "Current and future ferroelectric nonvolatile memory technology," *Journal of Vacuum Science and Technology B*, vol. 19, no. 5, pp. 1967–1971, 2001.

[16] D. Takashima, "Overview of FeRAMs: Trends and perspectives," *Non-Volatile Memory Technology Symposium (NVMTS), 2011 11th Annual*, pp. 1–6, Nov 2011.

[17] V. Rzehak, "Low-power fram microcontrollers and their applications," 2011. [Online]. Available: http://www.ti.com/lit/wp/slaa502/slaa502.pdf

[18] W. I. Kinney, W. Shepherd, W. Miller, J. Evans, and R. Womack, "A nonvolatile memory cell based on ferroelectric storage capacitors," *Electron Devices Meeting, 1987 International*, vol. 33, pp. 850–851, 1987.

[19] S. Eaton, D. Butler, M. Parris, D. Wilson, and H. McNeillie, "A Ferroelectric Nonvolatile Memory," *Solid-State Circuits Conference, 1988. Digest of Technical Papers. ISSCC. 1988 IEEE International*, pp. 130–, Feb 1988.

[20] K. Udayakumar, T. San, J. Rodriguez, S. Chevacharoenkul, D. Frystak, J. Rodriguez-Latorre, C. Zhou, M. Ball, P. Ndai, S. Madan, H. McAdams, S. Summerfelt, and T. Moise, "Low-power ferroelectric random access memory embedded in 180nm analog friendly CMOS technology," *Memory Workshop (IMW), 2013 5th IEEE International*, pp. 128–131, 2013.

[21] J. Rodriguez, J. Rodriguez-Latorre, C. Zhou, A. Venugopal, A. Acosta, M. Ball, P. Ndai, S. Madan, H. McAdams, K. Udayakumar, S. Summerfelt, T. San, and T. Moise, "180nm FRAM reliability demonstration with ten years data retention at 125°C," *Reliability Physics Symposium (IRPS), 2013 IEEE International*, pp. MY.11.1–MY.11.5, 2013.

[22] H. Zhang, J. Gummeson, B. Ransford, and K. Fu, "Moo: A batteryless computational RFID and sensing platform," Department of Computer Science, University of Massachusetts Amherst, Tech. Rep., 2011.

[23] B. Ransford, S. Clark, M. Salajegheh, and K. Fu, "Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling," *Proceedings of the 2008 conference on Power aware computing and systems*, pp. 5–5, 2008.

[24] V. Talla, B. Kellogg, B. Ransford, S. Naderiparizi, S. Gollakota, and J. R. Smith, "Powering the Next Billion Devices with Wi-Fi," *ACM CoNEXT*, 2015.

[25] W. S. Lee, H. Jayakumar, and V. Raghunathan, "When they are not listening: Harvesting power from idle sensors in embedded systems," *Green Computing Conference (IGCC), 2014 International*, pp. 1–10, Nov 2014.

[26] P. Zhang and D. Ganesan, "Enabling Bit-by-Bit Backscatter Communication in Severe Energy Harvesting Environments," *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp. 345–357, Apr. 2014.

[27] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient Backscatter: Wireless Communication out of Thin Air," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 39–50, Aug. 2013.

[28] P. A. Bernstein, *Concurrency control and recovery in database systems*. Addison-wesley New York, 1987, vol. 370.

[29] B. Lampson and H. Sturgis, *Crash recovery in a distributed data storage system*, 1979.

[30] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: transparent checkpointing under unix," *Proceedings of the USENIX 1995 Technical Conference Proceedings*, pp. 18–18, 1995.

[31] J. S. Plank, "An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance," Tech. Rep., 1997.

[32] S. I. Feldman and C. B. Brown, "IGOR: A System for Program Debugging via Reversible Execution," *SIGPLAN Not.*, vol. 24, no. 1, pp. 112–123, Nov. 1988.

[33] B. Ransford, "Transiently powered computers," Ph.D. dissertation, University of Massachusetts Amherst, 2013.

[34] B. Ransford, J. Sorber, and K. Fu, "Mementos: System Support for Long-running Computation on RFID-scale Devices," *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 159–170, 2011.

[35] M. Ahmadi and G. Jullien, "A Wireless-Implantable Microsystem for Continuous Blood Glucose Monitoring," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 3, no. 3, pp. 169–180, June 2009.

[36] D. Chen, Z. Liu, L. Wang, M. Dou, J. Chen, and H. Li, "Natural Disaster Monitoring with Wireless Sensor Networks: A Case Study of Data-intensive Applications upon Low-Cost Scalable Systems," *Mobile Networks and Applications*, vol. 18, no. 5, pp. 651–663, 2013.

[37] W. S. Lee, A. Kim, B. Ziaie, V. Raghunathan, and C. Powell, "UP-link: An ultra-low power implantable wireless system for long-term ambulatory urodynamics," *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*, pp. 384–387, Oct 2014.

[38] Y. Lee, S. Bang, I. Lee, Y. Kim, G. Kim, M. Ghaed, P. Pannuto, P. Dutta, D. Sylvester, and D. Blaauw, "A Modular 1 mm ^{3} Die-Stacked Sensing Platform With Low Power I ^{2} C Inter-Die Communication and Multi-Modal Energy Harvesting," *Solid-State Circuits, IEEE Journal of*, vol. 48, no. 1, pp. 229–243, Jan 2013.

[39] Atmel, "Innovative Techniques for Extremely Low Power Consumption with 8-bit Microcontrollers," http://www.atmel.com/images/doc7903.pdf, 2006.

[40] J. R. Levine, *Linkers and Loaders*. Morgen Kaufmann, 2000.

[41] Texas Instruments, "Msp430f543xa datasheet," http://www.ti.com/lit/ds/symlink/msp430f5438a.pdf.

[42] V. Saripalli, G. Sun, A. Mishra, Y. Xie, S. Datta, and V. Narayanan, "Exploiting Heterogeneity for Energy Efficiency in Chip Multiprocessors," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 1, no. 2, pp. 109–119, June 2011.

[43] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "TapeCache: A High Density, Energy Efficient Cache Based on Domain Wall Memory," *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 185–190, 2012.

[44] Tektronix, "Keithley 6430 Sub-Femtoamp Remote SourceMeter," http://www.keithley.com/products/dcac/sensitive/highresistance/?mn=6430.

[45] H. Jayakumar, A. Raha, and V. Raghunathan, "QuickRecall: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers," *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pp. 330–335, Jan 2014.

[46] H. Jayakumar, A. Raha, W. S. Lee, and V. Raghunathan, "QuickRecall: A HW/SW Approach for Computing Across Power Cycles in Transiently Powered Computers," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 1, pp. 8:1–8:19, Aug. 2015.

[47] D. Balsamo, A. Weddell, G. Merrett, B. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *Embedded Systems Letters, IEEE*, vol. 7, no. 1, pp. 15–18, March 2015.

[48] A. Rodriguez Arreola, D. Balsamo, A. K. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, and G. V. Merrett, "Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation," *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems*, pp. 3–8, 2015.

[49] F. A. Aouda, K. Marquet, and G. Salagnac, "Incremental checkpointing of program state to NVRAM for transiently-powered systems," *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014 9th International Symposium on*, pp. 1–4, May 2014.

[50] B. Cassens, A. Martens, and R. Kapitza, "The Neverending Runtime: Using New Technologies for Ultra-Low Power Applications with an Unlimited Runtime," *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, pp. 325–330, 2016.

[51] N. A. Bhatti and L. Mottola, "Efficient State Retention for Transiently-powered Embedded Sensing," *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, pp. 137–148, 2016.

[52] D. Balsamo, A. Weddell, A. Das, A. Arreola, D. Brunelli, B. Al-Hashimi, G. Merrett, and L. Benini, "Hibernus++: A Self-Calibrating and Adaptive System for Transiently-Powered Embedded Devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[53] C. Lu, V. Raghunathan, and K. Roy, "Micro-scale energy harvesting: A system design perspective," *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pp. 89–94, Jan 2010.

[54] ——, "Efficient Design of Micro-Scale Energy Harvesting Systems," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 1, no. 3, pp. 254–266, Sept 2011.

[55] C. Wang, N. Chang, Y. Kim, S. Park, Y. Liu, H. G. Lee, R. Luo, and H. Yang, "Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor," *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pp. 379–384, Jan 2014.

[56] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-case Execution-time Problem – Overview of Methods and Survey of Tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[57] B. Lucia and B. Ransford, "A Simpler, Safer Programming and Execution Model for Intermittent Systems," *SIGPLAN Not.*, vol. 50, no. 6, pp. 575–585, Jun. 2015.

[58] B. Ransford and B. Lucia, "Nonvolatile Memory is a Broken Time Machine," *Proceedings of the Workshop on Memory Systems Performance and Correctness*, pp. 5:1–5:3, 2014.

[59] A. Colin, A. P. Sample, and B. Lucia, "Energy-interference-free System and Toolchain Support for Energy-harvesting Devices," *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 35–36, 2015.

[60] A. Colin, G. Harvey, B. Lucia, and A. P. Sample, "An Energy-interference-free Hardware-Software Debugger for Intermittent Energy-harvesting Systems," *SIGOPS Oper. Syst. Rev.*, vol. 50, no. 2, pp. 577–589, Mar. 2016.

[61] M. Buettner, B. Greenstein, and D. Wetherall, "Dewdrop: An Energy-aware Runtime for Computational RFID," *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pp. 197–210, 2011.

[62] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue, "Fixing the Broken Time Machine: Consistency-aware Checkpointing for Energy Harvesting Powered Non-volatile Processor," *Proceedings of the 52nd Annual Design Automation Conference*, pp. 184:1–184:6, 2015.

[63] L. Kothari and N. Carter, "Architecture of a Self-Checkpointing Microprocessor that Incorporates Nanomagnetic Devices," *Computers, IEEE Transactions on*, vol. 56, no. 2, pp. 161–173, Feb 2007.

[64] W. Yu, S. Rajwade, S.-E. Wang, B. Lian, G. Suh, and E. Kan, "A non-volatile microcontroller with integrated floating-gate transistors," *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pp. 75–80, 2011.

[65] N. Onizawa, A. Mochizuki, A. Tamakoshi, and T. Hanyu, "A sudden power-outage resilient nonvolatile microprocessor for immediate system recovery," *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*, pp. 39–44, July 2015.

[66] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 526–537, Feb 2015.

[67] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan, "Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications," *IEEE Micro*, vol. 35, no. 5, pp. 32–40, Sept 2015.

[68] X. Sheng, Y. Wang, Y. Liu, and H. Yang, "SPaC: A Segment-based Parallel Compression for Backup Acceleration in Nonvolatile Processors," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013, pp. 865–868.

[69] M. Zhao, Q. Li, M. Xie, Y. Liu, J. Hu, and C. J. Xue, "Software assisted non-volatile register reduction for energy harvesting based cyber-physical system," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, 2015, pp. 567–572.

[70] Q. Li, M. Zhao, J. Hu, Y. Liu, Y. He, and C. J. Xue, "Compiler Directed Automatic Stack Trimming for Efficient Non-volatile Processors," in *Proceedings of the 52nd Annual Design Automation Conference*. New York, NY, USA: ACM, 2015, pp. 183:1–183:6.

[71] M. Xie, C. Pan, J. Hu, C. Yang, and Y. Chen, "Checkpoint-aware instruction scheduling for nonvolatile processor with multiple functional units," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 316–321.

[72] Z. Li, Y. Liu, D. Zhang, C. J. Xue, Z. Wang, X. Shi, W. Sun, J. Shu, and H. Yang, "HW/SW Co-design of Nonvolatile IO System in Energy Harvesting Sensor Nodes for Optimal Data Acquisition," *Proceedings of the 53rd Annual Design Automation Conference*, pp. 154:1–154:6, 2016.

[73] T. K. Chien, L. Y. Chiou, C. C. Lee, Y. C. Chuang, S. H. Ke, S. S. Sheu, H. Y. Li, P. H. Wang, T. K. Ku, M. J. Tsai, and C. I. Wu, "An energy-efficient nonvolatile microprocessor considering software-hardware interaction for energy harvesting applications," *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–4, April 2016.

[74] H. Qin, "Deep Sub-Micron SRAM Design for Ultra-Low Leakage Standby Operation," Ph.D. dissertation, EECS Department, University of California, Berkeley, May 2007.

[75] J. Wang and B. Calhoun, "Canary Replica Feedback for Near-DRV Standby VDD Scaling in a 90nm SRAM," *Custom Integrated Circuits Conference, 2007. CICC '07. IEEE*, pp. 29–32, Sept 2007.

[76] J. Kulkarni, K. Kim, S. P. Park, and K. Roy, "Process variation tolerant SRAM array for ultra low voltage applications," *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pp. 108–113, June 2008.

[77] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pp. 148–157, 2002.

[78] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, Feb 2003.

[79] T. Houston, "SRAM device and a method of operating the same to reduce leakage current during a sleep mode," *US Patent 7,307,907*, Dec. 11 2007.

[80] M. Lysinger, D. McClure, and F. Jacquet, "Sram with switchable power supply sets of voltages," *US Patent 7,623,405*, Nov. 24 2009.

[81] D. Holcomb, W. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *Computers, IEEE Transactions on*, vol. 58, no. 9, pp. 1198–1210, Sept 2009.

[82] Texas Instruments, "Msp430g2x52 datasheet," http://www.ti.com/lit/ds/symlink/msp430g2452.pdf.

[83] S. Mukhopadhyay, H. Mahmoodi-Meimand, and K. Roy, "Modeling and estimation of failure probability due to parameter variations in nano-scale SRAMs for yield enhancement," *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pp. 64–67, June 2004.

[84] J. Borgeson, "Ultra-low-power pioneers: TI slashes total MCU power by 50 percent with new Wolverine MCU platform," http://www.ti.com/ww/en/mcu/wolverine/wolverine-whitepaper.pdf, February 2012.

[85] E. Vatajelu and J. Figueras, "Statistical analysis of 6T SRAM data retention voltage under process variation," *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on*, pp. 365–370, April 2011.

[86] H. Le, N. Fong, and H. Luong, "RF energy harvesting circuit with on-chip antenna for biomedical applications," *Communications and Electronics (ICCE), 2010 Third International Conference on*, pp. 115–117, Aug 2010.

[87] T. Le, K. Mayaram, and T. Fiez, "Efficient Far-Field Radio Frequency Energy Harvesting for Passively Powered Sensor Networks," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 5, pp. 1287–1302, May 2008.

[88] H. Visser and R. Vullers, "RF Energy Harvesting and Transport for Wireless Sensor Network Applications: Principles and Requirements," *Proceedings of the IEEE*, vol. 101, no. 6, pp. 1410–1423, June 2013.

[89] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient Backscatter: Wireless Communication out of Thin Air," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 39–50, Aug. 2013.

[90] Y. Chu, L. Wan, G. Ding, P. Wu, D. Qiu, J. Pan, and H. He, "Flexible ZnO nanogenerator for mechanical energy harvesting," *Electronic Packaging Technology (ICEPT), 2013 14th International Conference on*, pp. 1292–1295, Aug 2013.

[91] Z. Lin Wang, X. Wang, J. Song, J. Liu, and Y. Gao, "Piezoelectric Nanogenerators for Self-Powered Nanodevices," *Pervasive Computing, IEEE*, vol. 7, no. 1, pp. 49–55, Jan 2008.

[92] J. Liu, P. Fei, J. Zhou, R. Tummala, and Z. L. Wang, "Toward high output-power nanogenerator," *Applied Physics Letters*, vol. 92, no. 17, 2008.

[93] Maxim Integrated Products Inc., "MAX4645-Fast, Low-Voltage, 2.5Ω, SPST, CMOS Analog Switches," http://datasheets.maximintegrated.com/en/ds/MAX4645-MAX4646.pdf.

[94] Analog Devices, "ADG819, 0.5 Ohms, CMOS, 1.8 V to 5.5 V, 2:1 Mux/SPDT Switch," http://www.analog.com/static/imported-files/data_sheets/ADG819.pdf.

[95] M. Gruhn and T. Muller, "On the Practicability of Cold Boot Attacks," *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pp. 390–397, 2013.

[96] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest We Remember: Cold-boot Attacks on Encryption Keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, May 2009.

[97] N. Saxena and J. Voris, "We Can Remember It for You Wholesale: Implications of Data Remanence on the Use of RAM for True Random Number Generation on RFID Tags," *CoRR*, vol. abs/0907.1256, 2009.

[98] S. Skorobogatov, "Low temperature data remanence in static RAM," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-536, Jun. 2002.

[99] D. Holcomb, A. Rahmati, M. Salajegheh, W. P. Burleson, and K. Fu, "DRV-Fingerprinting: Using Data Retention Voltage of SRAM Cells for Chip Identification," *The 8th Workshop On RFID Security And Privacy*, Jul. 2012.

[100] A. Rahmati, M. Salajegheh, D. Holcomb, J. Sorber, W. P. Burleson, and K. Fu, "TARDIS: Time and Remanence Decay in SRAM to Implement Secure Protocols on Embedded Devices without Clocks," *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pp. 221–236, 2012.

[101] U. Kulau, F. Büsching, and L. Wolf, "IdealVolting: Reliable Undervolting on Wireless Sensor Nodes," *ACM Trans. Sen. Netw.*, vol. 12, no. 2, pp. 11:1–11:38, Apr. 2016.

[102] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting Partially-Forgetful Memories for Approximate Computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, March 2015.

[103] H. Jayakumar, A. Raha, and V. Raghunathan, "Hypnos: An Ultra-low Power Sleep Mode with SRAM Data Retention for Embedded Microcontrollers," *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, pp. 11:1–11:10, 2014.

[104] ——, "Sleep Mode Voltage Scaling: Enabling SRAM Data Retention at Ultra-Low Power in Embedded Microcontrollers," *ACM Transactions on Embedded Computing Systems*, 2016.

[105] Texas Instruments, "TMP20 Analog Temperature Sensor." [Online]. Available: http://www.ti.com/lit/ds/symlink/tmp20.pdf

[106] ONSemi, "NCP302 Voltage Detector." [Online]. Available: http://www.onsemi.com/pub_link/Collateral/NCP302-D.PDF

[107] Texas Instruments, "TPS22901: Low Input Load Switch." [Online]. Available: http://www.ti.com/lit/ds/symlink/tps22901.pdf

[108] BlueGiga, "BLE113 SoC Module." [Online]. Available: https://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/ble113-bluetooth--smart-module/technical-specs/

[109] Maxim IC, "Maxim 4652 SPST switches." [Online]. Available: http://datasheets.maximintegrated.com/en/ds/MAX4651-MAX4653.pdf

[110] Intersil, "ISL84715 SPST Switch." [Online]. Available: http://www.intersil.com/content/dam/Intersil/documents/isl8/isl84715-16.pdf

[111] Monsoon Power Solutions, "Monsoon power monitor." [Online]. Available: http://msoon.github.io/powermonitor/PowerTool/doc/Power%20Monitor%20Manual.pdf

[112] Tektronix, "Tektronix MDO4104-3." [Online]. Available: http://www.tek.com/oscilloscope/mdo4000-mixed-domain-oscilloscope-manual/mdo4000b-series

[113] Saleae, "Saleae logic analyzer." [Online]. Available: https://www.saleae.com/

APPENDIX

APPENDIX
QUBE: AN FeRAM-BASED, LOW POWER, MODULAR PLATFORM ARCHITECTURE FOR INTERMITTENTLY-POWERED IoT DEVICES

This appendix describes the design and power consumption characteristics of an IoT platform, QUBE, that was used for evaluating QuickRecall in Chapter 3 and the later techniques proposed in Chapter 4.

A.1 Introduction

Various industry forecasts project that, by 2020, there will be around 50 billion devices [1] connected to the Internet of Things (IoT), helping to engineer new solutions to a variety of societal-scale problems such as health-care, energy conservation, transportation, *etc.* Most of these devices will be wireless due to the expense, inconvenience, or in some cases, the sheer infeasibility of wiring them. Further, many of them will have stringent size constraints. With no cord for power and limited space for a battery, powering these devices (to achieve several months to possibly years of unattended operation) becomes a daunting challenge [2, 3]. Therefore, designing ultra low power platforms with minuscule amount of sleep mode power consumption is crucial to the success and widespread adoption of the IoT vision.

Recent advances in semiconductor technology have resulted in the emergence of memory technologies such as Ferroelectric RAM (FeRAM), Magnetoresistive RAM (MRAM), *etc.,* that combine the speed, flexibility, and endurance of SRAM with the non-volatility of flash, all at a very low power consumption. This work utilizes QuickRecall [45] that utilize the emerging NVM as both the RAM and ROM of the

system to reduce checkpointing and restore overheads. Thus, QUBE makes the case that the use of these emerging memories, with better power performance characteristic than flash, significantly advances the state-of-the-art in ultra-low power computing platforms for IoT edge devices. Specifically, our contributions are listed below:

- We propose QUBE, a novel FeRAM-based hardware platform for IoT edge devices that enables ultra-low power operation. QUBE utilizes a modular architecture, which is composed of separate 1 `inch` $\times$ 1 `inch` modules stacked together in a plug-and-play manner.
- We define a generic bus architecture, QUBUS, that binds the constituent modules of QUBE together. Adherence to a fixed bus architecture expedites the design, development, and testing of the individual modules and enables seamless integration of the complete IoT edge device.
- QUBE features several low power optimizations, such as individual power domains for modules. The power domains can be independently disabled, which allows for module-level power gating of the system. Also, QUBE's use of FeRAM, instead of SRAM, means that the contents of memory are preserved *in-situ* across power cycles, avoiding the need for (and the energy overhead of) backing them up to persistent storage, such as flash, when power loss is imminent. Using these low power features, we demonstrate a typical wireless sensing application on QUBE that consumes only 4 μA in sleep mode.

A.2 Hardware architecture

The design goal for QUBE is to create an ultra-low power platform that is modular and has a small form factor. In this section, we first describe the modular QUBE design, followed by the bus architecture that enables it. Finally, we explain the design decisions made to enable the ultra-low power operation of QUBE.

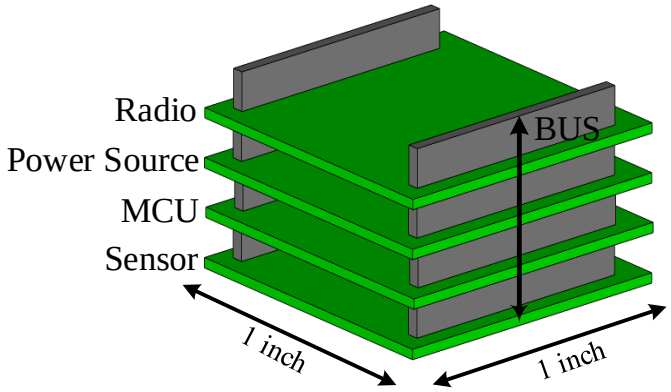


Fig. A.1. QUBE Modular Architecture

A.2.1 Modular design

An IoT edge device is typically composed of multiple functional sub-systems such as a computation sub-system, a radio sub-system, a sensor sub-system, a power sub-system, *etc.* In the QUBE design, this functional demarcation is also adopted in the physical sense to create an implementation that places each functional sub-system on a physically separate module. Fig. A.1 illustrates the proposed QUBE architecture wherein four such modules are placed in a stack and connected using a common bus. The primary advantage of such an architecture is the effortless customizability of the entire system. First, the modular architecture offers easy addition and removal of features to the edge device. For example, a designer wishing to work with a Zigbee radio instead of a Bluetooth radio can simply replace the radio module in the stack. In another case, a designer who wants to add a new sensor can just plug in an additional sensor module to the stack. Second, the architecture indirectly assists in accelerating the design, debug, and prototyping phases involved in the development of the entire system. A modular design, such as QUBE, will enable designers to focus on the development of each module as an independent entity. Each module can then be tested separately in an isolated environment without interference from other modules. Once each module is tested, the entire system can be integrated by stacking the

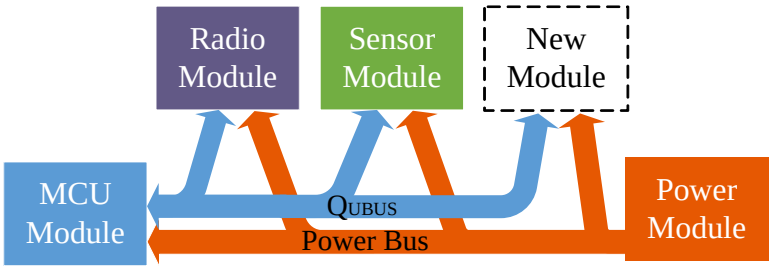


Fig. A.2. The QUBE interconnect topology

Table A.1.
The QUBUS Architecture

Function	#Ports	Bus Channels
Power Enable	4	4
UART	1	4
Analog Channels	4	4
Interrupt Capable GPIOs	4	4
I2C	1	2
SPI	1	7
GPIO & Auxiliary ports	-	12

modules and be subjected to further verification and debugging. Therefore, QUBE's design philosophy is aligned with the logical flow of system integration and testing that hardware engineers typically adhere to.

A.2.2 The QUBE interconnect

A key feature that enables the QUBE modular architecture is the interconnect topology. The interconnect structure in QUBE has two parts, namely, a general purpose bus (QUBUS) and a power bus. Fig. A.2 illustrates the QUBE interconnect topology and Table A.1 details the QUBUS architecture.

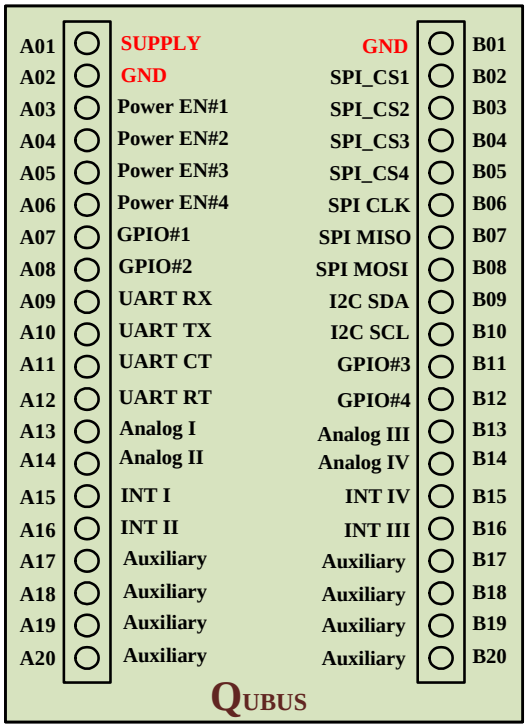


Fig. A.3. Power bus and QUBUS on headers

QUBUS Architecture: The QUBUS is a bidirectional bus connecting the MCU module to other modules. Its primary function is to enable communication between various system components. The QUBUS architecture specifies a minimum set of peripheral functions to be brought out from the MCU module. The QUBUS supports four different power domains with dedicated power enable signals. It also contains interfaces for typical serial communication protocols such as UART (4-wire), I2C, and SPI (with 4 different chip selects). The QUBUS also has a minimum of 4 analog channels and 4 GPIOs with interrupt capabilities. Additionally, 4 GPIOs are brought out to the bus from the MCU that may be multiplexed with other peripheral functionality as the designer may deem fit. Finally, 8 channels are left unused on the bus to facilitate ease of debugging, make application-specific assignments, and allow for future expansion.

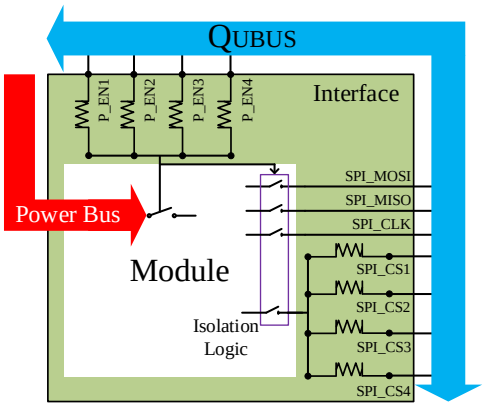


Fig. A.4. Bus interface on a QUBE module

The interconnect is implemented as two rows of 20-pin headers on each module as shown in Fig. A.3. The headers propagate the signals across the layers of the stack. Adhering to such a fixed bus architecture simplifies the task of designing a new module. For example, if a new sensor module wants to use the I2C bus, the designer can simply route to the I2C channels of the QUBUS. To avoid multiple slaves using the same bus channel, it is important to define a module to QUBUS interface on each module. The QUBE design places a constraint on the designer to adhere to the defined module to bus interface for select functions. For example, the bus interface for enabling SPI on a module is shown in Fig. A.4. The SPI clock and data lines go directly into the module. However, all the four SPI chip selects on the QUBUS are brought onto the interface of the module. During system integration, one of the four chip select paths is closed using a 0 Ω resistor and fed into the module. Thus, the designer can decide upon the chip select to use at deployment time and avoid potential conflicts with other SPI-enabled modules with minimal change in software. Note that each SPI slave on the QUBUS needs to replicate this interface.

Power Bus: The power bus consists of 3 channels, *i.e.,* 2 ground lines and 1 power supply line. It originates from the power module and supplies power to the entire QUBE. The power bus also has a power-enable interface in each module that is discussed next.

A.2.3 Ultra-low power design

To enable ultra-low power consumption, QUBE supports a maximum of five different power domains; namely, the main power supply domain and four domains controlled by the power enable bits of the QUBUS. As illustrated in Fig. A.4, each module has a power domain interface similar to the SPI chip select interface that connects the correct power enable to the module. The selected power enable is fed into a switch that gates the power supply to the module. Such an architecture serves two purposes. First, it grants a system designer the freedom to effortlessly implement and allocate power-domains. The designer could consider each discrete module to be an independent power-domain oblivious to the control required at a system level. Then, during the stack assembly, he/she can allocate the appropriate power domain to each module. Second, since all the peripherals and components associated with a module (such as pull-up resistors, decoupling capacitors, *etc.*) are located on the module itself, the low power mode implementation becomes much more efficient as none of these components consume power when the entire module is power gated.

To maintain signal integrity and to prevent unwanted leakage or capacitive loading on bus channels that are used by multiple modules on the stack, a bus isolation interface is defined. The function of the interface is to provide isolation for the connected channels and thereby prevent any floating lines when the module is in power-gated mode. The isolation must be done universally for the channel across the system. QUBE achieves isolation using simple analog switches controlled by the particular module's power enable signal. Therefore, when a module is powered on, all its bus channels are active as well. Thus, multiple power domains can be controlled in this fashion to optimize the power consumption. The following describes the power management of a typical wireless sensing QUBE.

The MCU controls the power supply of the other modules depending on the application task. For example, a wireless sensor's activities can be broken down into the atomic operations of sense, store, send, and sleep, which are often repetitive in

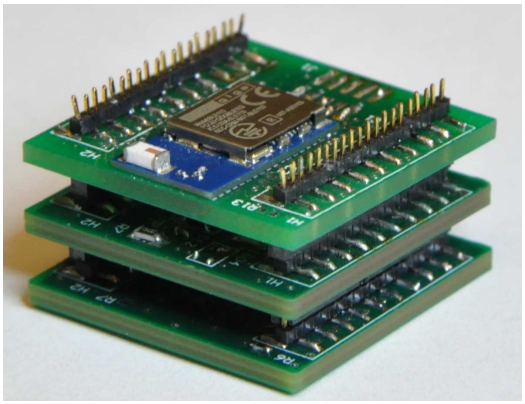
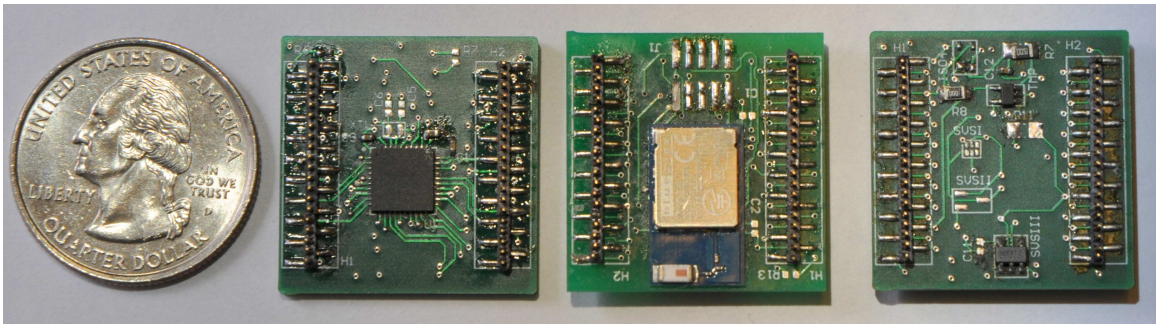


Fig. A.5. QUBE Stack



(a) MCU Module (b) Radio Module (c) Sensor Module

Fig. A.6. QUBE Functional Modules

nature. All these operations have varying power requirements. Therefore, on waking up, only the MCU module needs to be powered on. During a sense operation, the MCU wakes up the sensor module. For the ensuing transmit operation, the MCU and radio module need to be powered on. Therefore, only the relevant modules can be supplied with power depending on the task currently being executed.

A.3 Low power implementation

To evaluate the QUBE architecture, we designed and fabricated an MCU module, a sensor module, and a radio module. Fig. A.6 shows our implementation of each module separately, and Fig. A.5 shows the stacked QUBE. For uniformity, the size of each module is 25 mm x 25 mm and the height of each module is bounded by a post length of 4 mm.

The MCU module consists of the TI MSP430FR5739 microcontroller [10] that has 16 KB of FeRAM. The sensor module consists of an analog temperature sensor, TMP20 from TI [105], and a power management unit (PMU). The PMU consists of a supply voltage supervisor (SVS), NCP302 [106], and a power switch, TPS22901 from TI [107]. The radio module consists of a bluetooth low energy SoC module, BLE113 from BlueGiga [108]. Additionally, the signal isolation and power gating on each module are provided by ultra-low quiescent current SPST switches (MAX4652 [109] and ISL84715 [110]).

The QUBE architecture facilitates easy allotment of power domains according to modules. Thus, two power domains are defined for QUBE, namely, the radio domain and the sensor domain. The MCU receives power from the source directly and routes power to the two power domains depending on the application state. For example, the BLE module receives power only when the application is required to transmit data. This modular power management technique allows us to switch off the unused components according to functionality, thus enabling ultra low power operation.

As Fig. A.5 shows, the modules are stacked vertically and are connected through the headers. The BLE module occupies the top-most layer in order to reduce antenna interference. It is programmed as a slave and interacts with the MCU master module through the UART interface on the QUBUS. The BLE is programmed to transmit at its lowest power configuration of -24 dbm. Lastly, the sensor module uses an analog channel on the QUBUS to interface with the MCU.

A.4 Example usage scenario

Intermittently-powered devices are a new class of batteryless IoT devices that receive energy from unreliable power sources. Therefore, it often receive power in intermittent bursts. To enable computations across these power cycles is a challenge for such systems. The target application needs to store data pertaining to the program and processor states in a non-volatile memory before power is lost so that the state can be recalled after a subsequent power up. QuickRecall [45] is a lightweight, in-situ checkpointing technique using FeRAM that seamlessly enables long-running computations in intermittently-powered systems.

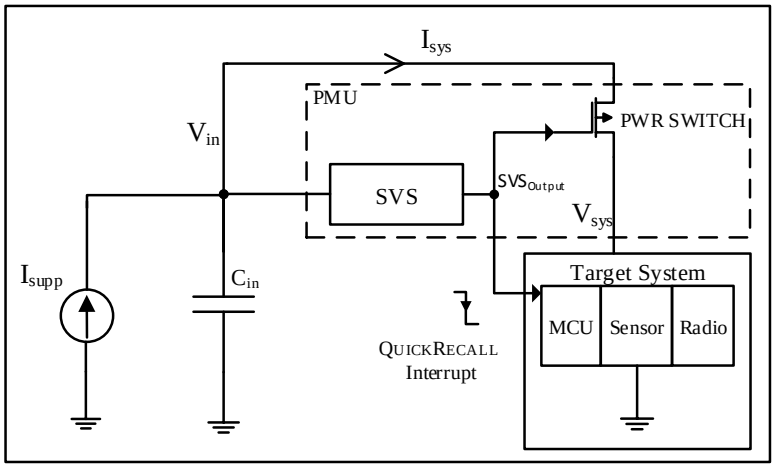


Fig. A.7. QUBE setup for enabling intermittently-powered systems

QuickRecall is unlike conventional checkpointing schemes that are either periodic in nature or are initiated by inserting appropriate triggers at vantage locations in the program. QuickRecall performs a checkpoint operation only when it detects that the supply voltage is below a critical operating threshold voltage. Such a checkpointing scheme does not impede normal program execution and only triggers a checkpoint if power loss is imminent. Fig. A.7 shows the setup required for implementing QuickRecall on Qube. An external SVS is used to monitor the supply voltage, V_{in}, and

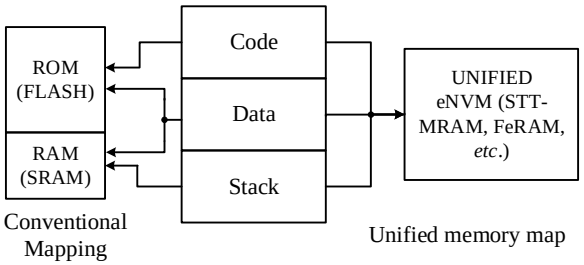


Fig. A.8. QuickRecall Linker Map

trigger the MCU interrupt when it falls below the SVS_{OFF} threshold. Alternatively, it closes the power switch when $\text{V}_{in} \geq \text{SVS}_{ON}$ and supplies power to the target system.

The interrupt triggers a checkpoint operation that saves the system context. The system context consists of program state, processor state, and the state of configuration registers of various peripheral subsystems. Each of the above mentioned state information has to be retained for a successful recall and resumption of computation across power cycles. Fig. A.8 shows the linker map proposed by QuickRecall that uses an NVM technology such as FeRAM as unified memory. Conventionally, the linker maps the code section to a non-volatile storage like flash, and the `data`, `bss`, and `stack` sections to the volatile SRAM. The same non-volatile memory is partitioned by the linker to include all the sections. The non-volatile memory now acts as the conventional RAM as well as the ROM. As a result, while the MCU powers off, the RAM data is saved in-situ. Similarly, while waking up, the program can pick up the data from exactly the same address locations. By using FeRAM as the RAM, QuickRecall is superior to previous checkpointing schemes as there is no time or energy overhead incurred to retain RAM data. Processor state denotes the state of the microcontroller register file, which includes the program counter (PC), stack pointer (SP), status register (SR), and General Purpose Registers (GPRs). QuickRecall saves the values of these registers during checkpointing. Finally, it saves the configuration registers for the MCU and associated peripherals onto the FeRAM as per application requirement and recalls the state in the subsequent boot sequence.

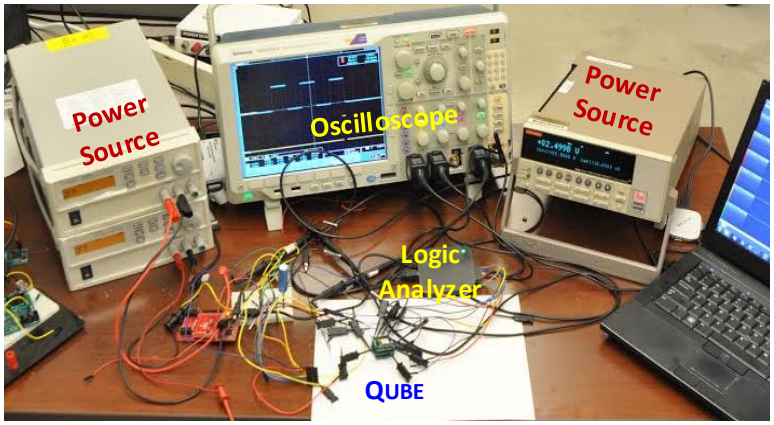


Fig. A.9. Experimental setup

A.5 Evaluation

This section describes our experimental setup and evaluation of QUBE.

A.5.1 Experimental setup

Fig. A.9 shows the experimental setup for QUBE. For all the experiments, the system was supplied with power using DC voltage and current sources. Power was measured using a Tektronix 6430 Keithley source meter [44], which can measure up to femtoamperes, and a Monsoon Power Monitor [111]. A Tektronix MDO4104-3 oscilloscope [112] was used to measure the latency overheads. Lastly, the Saleae logic analyzer [113] was used to snoop the QUBUS.

A.5.2 QUBE power measurements

In this experiment, QUBE's power consumption is characterized. A constant power source is used to supply power to the system. A simple program that was used for evaluation is described below. Initially, only the microcontroller module is powered on and all the other power domains are shutdown. Then, the microcontroller activates the sensor module and samples temperature data from the sensor. Once the sampling

Table A.2.
Stand-alone current consumption

Mode	**Current Consumption (mA)**	**Execution time (ms)**
Sense	0.51	1.167
Compute	0.3	0.453
Transmit (−24 dbm)	18	330.8
Idle	0.004	-

is done, the sensor module is turned off. The samples are then averaged and the average is subjected to an encryption step. Subsequently, the BLE power domain is enabled and the encrypted data is packed and advertised[1] using the Bluetooth radio. Finally, the BLE power domain is cut-off and the microcontroller enters low power mode.

Table A.2 shows the current consumption and execution time of each mode. The measurements are made in steady-state after the entire system is powered on. The `sense` and `compute` operations are done on 10 samples of sensor data, and the transmit time noted is the time taken to advertise a single packet. `Sense` consumes more power than `compute` due to the ADC being utilized for conversion.

Fig. A.10 shows the current consumption of QUBE for each atomic task in a sense-and-send application with a supply voltage of 2.6 V. Note that we intentionally collect more samples, perform computations, and transmit multiple times in order to make a visual distinction between the current consumption of the different modes. 2048 samples are collected for performing `sense` and `compute` operations. After `compute` operation is completed, the radio power domain is enabled for transmission, which leads to an in-rush current that charges up the capacitors of the BLE module. About

[1]Advertising does not require bluetooth pairing to occur.

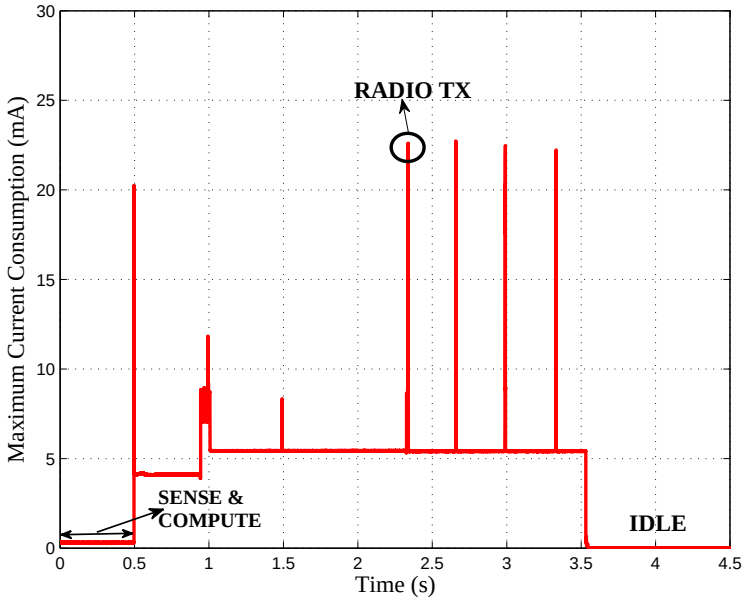
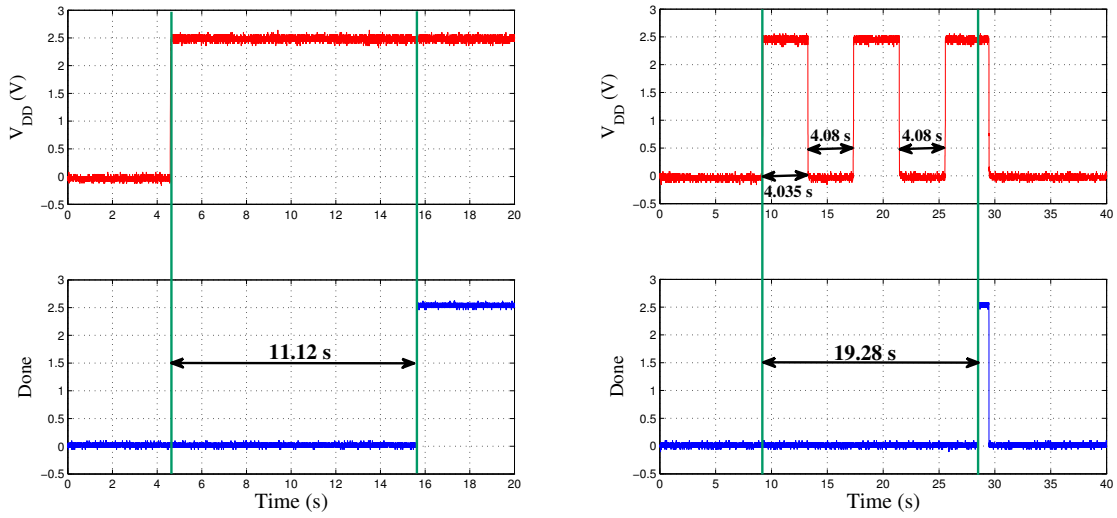


Fig. A.10. QUBE current consumption trace

500 ms later, the BLE begins its boot process, which is denoted by the second spike at 1 s. The BLE then enters active mode and transmits two times. Each time two different packets are advertised: a normal packet and the sensor data packet leading to four transmission peaks as shown in the figure. Then the system enters idle mode wherein all the power domains are disabled and the MCU enters the lowest power mode. Note that during idle time, the current consumption is as low as 4 μA. Only the components in the power supply domain contribute to the idle power, and this primarily includes the PMU and associated logic for power supply management.

A.5.3 Computing across power cycles

In the usage scenario described in Section A.4, the goal of QUBE is to enable computations seamlessly across power cycles. To demonstrate this property, two experiments are performed. The evaluation application for the experiments performs a 64-bit RSA encryption program on 128 different characters. A "`Done`" signal indicates



(a) Continuous power supply (b) Intermittent power supply

Fig. A.11. Execution of RSA encryption on QUBE

the successful completion of encryption and is brought out on a GPIO channel of the QUBUS. The experiments are described below.

For the first experiment, the supply voltage is provided *continuously* and is kept at a constant 2.5 V. Fig. A.11(a) shows the supply voltage for QUBE and the `Done` signal for the application. Observe that the `Done` signal is set 11.12 s after receiving the power supply, which is the application's execution time. In the second experiment, QUBE is supplied using an intermittent power source (that power-cycles three times), and the total time required for application execution is measured. Fig. A.11(b) shows the power-cycle operation on the supply voltage and the `Done` signal. The duration of a single power cycle is 4.035 s, and successive power cycles are separated by an `OFF` duration of 4.08 s as denoted in the figure. Therefore, a single power cycle is insufficient for completing the program. However, observe that the done signal is raised in the third cycle, which proves that the program computed successfully across power cycles.

Table A.3.
Program Exec. Time (CPU Freq = 8 MHz)

Program	**Overhead**[a]	**Total Exec. Time**
RSA	12.06 μs + 580 μs + 9 μs	11.12 s
CRC	12.06 μs + 580 μs + 9 μs	547 ms
SENSE	12.06 μs + 17.6 ms + 9 μs	73 ms

[a] Store Overhead + Initialization Overhead + Restore Overhead

Additionally, the QuickRecall overhead per power cycle and the single power cycle execution time for three different programs are tabulated in Table A.3. The overhead comprises of constant store and restore overheads of 12.06 μs and 9 μs respectively, and an initialization overhead that varies from one application to another. The initialization overhead is defined as the time required for the program to configure the MCU and enable the necessary peripherals. For `RSA`, note that the QuickRecall overhead is in the order of microseconds. Therefore, the overhead for the completing the application across three power cycles is negligible, which concurs with our result (19.28 s$-$ 2 $\times$ 4.08 s = 11.12 s) as shown in Fig. A.11. The `sense` application requires the sensor and ADC to settle before an accurate reading can be made. Therefore, the initialization overhead is relatively longer than that of the other programs. For all the three programs, QUBE was able to implement QuickRecall and successfully compute across power cycles.

A.6 Conclusions

In this design, we have demonstrated QUBE, a generic low power modular architecture for IoT edge devices that consumes only 4 μA in idle mode. A new bus architecture, QUBUS, is defined that facilitates modular development, testing and

integration of the constituent modules of an embedded system. QUBE is generic and allows effortless addition and removal of features by its plug-n-play architecture. Finally, using QUBE, we demonstrated successful and seamless computation across power cycles with negligible overhead in an intermittently-powered IoT device.

VITA

VITA

Hrishikesh Jayakumar received the B.Tech. degree in Electrical Engineering and the M.Tech. degree in Microelectronics and VLSI Design (Dual Degree) from the Indian Institute of Technology, Madras in 2009. After having worked in the industry for a year, he has been pursuing his Ph.D. degree in Electrical and Computer Engineering at Purdue University, West Lafayette, USA.

Mr. Jayakumar's current research interests include hardware and software architectures for embedded systems and the Internet of Things with a particular focus on low-power design and reliability, emerging memory technologies, and approximate computing. His industry experience includes an internship with the system architecture team Qualcomm Inc., San Diego, USA, in the summer of 2015, as an SoC Design Engineer at Qualcomm Inc., Bangalore, India, from 2009 to 2010, and as a CAD software intern at Texas Instruments, Chennai, India in the summer of 2007. Mr. Jayakumar has served as an expert reviewer for ACM Transactions on Sensor Networks (ToSN) in 2015 and 2016, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD) in 2015, and ACM/IEEE Design Automation Conference (DAC) in 2015.

Mr. Jayakumar received the best paper award at the IEEE International Conference on VLSI Design in 2016 and has won design contest awards at the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED) in 2014, and at the IEEE International Conference on VLSI Design in 2015.